VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYHLEDÁVÁNÍ INFORMACÍ V ČESKÉ WIKIPEDII

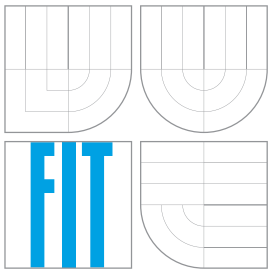
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK BALGAR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYHLEDÁVÁNÍ INFORMACÍ V ČESKÉ WIKIPEDII

INFORMATION RETRIEVAL IN CZECH WIKIPEDIA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MAREK BALGAR
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR CHMELAŘ
SUPERVISOR

BRNO 2011

Abstrakt

Tato diplomová práce má za úkol proniknout do problematiky vyhledávání informací a klasifikace textu. Hlavní výzkum se zde zaměřuje na textová data, sémantické slovníky a hlavně na znalosti odvoditelné z encyklopedií jako je Wikipedie. V této práci je dále popsaná implementace systému pro dotazování, který je vytvořen na základě získaných znalostí. V závěru práce jsou zhodnoceny vlastnosti a případná vylepšení vyhledávače.

Abstract

The main task of this Masters Thesis is to understand questions of information retrieval and text classification. The main research is focused on the text data, the semantic dictionaries and especially the knowledges inferred from the Wikipedia. In this thesis is also described implementation of the querying system, which is based on achieved knowledges. Finally properties and possible improvements of the system are talked over.

Klíčová slova

Vyhledávání infomací, index term, koncepty, sémantika, Wikipedie, vektorový model, Thezaurus, sémantické slovníky, textová data, pravděpodobnostní model, booleovský model, stemmer, relevantní informace, indexace, stop slova, filtrování, dolování dat

Keywords

Information retrieval, index term, concepts, semantics, Wikipedia, vector model,Thezaurus, semantic dictionary, text data, probablistic model, boolean model, stemmer, relevant information, indexing, stop words, filtering, data mining

Citace

MAREK BALGAR: VYHLEDÁVÁNÍ INFORMACÍ V ČESKÉ WIKIPEDII, diplomová práce, Brno, FIT VUT v Brně, 2011

VYHLEDÁVÁNÍ INFORMACÍ V ČESKÉ WIKIPEDII

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Petra Chmelaře

.....
MAREK BALGAR
25. května 2011

Poděkování

Děkuji svému vedoucímu Ing. Petru Chmelařovi za odborné vedení a podněty, které mi při řešení tohoto projektu poskytl.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Už od počátku civilizace se lidstvo snažilo zdokonalit předávání a uchovávání informací. Postupem času se tak zdokonalovaly techniky, které umožňovaly snadnější vyjádření a uchování myšlenek. Velkým zlomem tak byl bezesporu vznik strojově čitelných medii, což umožnilo přenositelnost dat na velice malých plochách. Nemuselo se pak provádět prohledávání celé knihovny, protože vše bylo na jednom místě. Tento přístup byl ale stále nedokonalý, neboť vyhledávání založené na postupném procházení jednotlivých položek zabíralo neadekvátní množství času. Bylo proto třeba najít nástroje, pomocí kterých je možné efektivní a rychlé vyhledávání informací ve velkých objemech dat. Společně se zdokonalováním strojově čitelných medii tak byly v 50. a 60. letech 20. století vyvíjeny první systémy pro automatické vyhledávání informací. V dnešní době je tak hlavním požadavkem vytvářet takové vyhledávače, které budou rychlé, efektivní a s požadovanými výsledky.

Úkolem vyhledávání informací je poskytnout uživateli, po zadání dotazu, dokumenty, které se snaží vyhledat. Kvalitu systémů pro vyhledávání informací lze hodnotit více způsoby. Většinou hodnotíme, jestli dokumenty nabídnuté uživateli jsou skutečně relevantní a jaké množství relevantních dokumentů byl algoritmus schopen nalézt.

Cílem této práce je neprve proniknout do problematiky vyhledávání informací a klasifikace textu. Můj výzkum se tak zaměřil na textová data, sémantické slovníky a hlavně na znalosti odvoditelné z encyklopedii jako je Wikipedie. Dalším krokem bude nastudování aktuálních problémů v oblasti vyhledávání informací a jejich případné vyřešení. Zde se zaměřím hlavně na vyhledávání v rámci interaktivně specifikovaných kontextů. Další části pak bude vytvořit systém pro dotazování, který bude schopen kvalitně a rychle vyhledávat. Co se týče implementace, tak mým úkolem bude zaměřit se hlavně na vyhledávání v české wikipedii. Posledním krokem pak bude zhodnocení vlastností a případných vylepšení vyhledávače.

V druhé kapitole této práce se podrobněji rozepíši o problematice vyhledávání informací. Postupně zde vysvětlím základní pojmy, rozdíl mezi vyhledáváním informací a datovým vyhledáváním.

V třetí kapitole popíši jednotlivé principy vyhledávání, které se používají na vyhledávání textových informací. Rozeberu zde jejich hlavní principy, klady a zápory. Nakonec provedu jejich porovnání a samozřejmě zhodnocení.

Ve čtvrté kapitole detailněji uvedu jednotlivé reprezentace dat pro vyhledávání informací. Zaměření zde tedy bude na textová data, sémantické slovníky a encyklopedie.

V páté kapitole bude okomentována a rozebrána problematika získávání informací z Wikipedie. Popíši zde způsob, jak se z ní získávají informace a také jak se provádí vyhodnocování interaktivně specifikovaných kontextů. Hlavní zaměření je zde na strukturu databáze

a význam jednotlivých tabulek. Další důležitou částí, která je zde detailně popsána je Wikipedia Miner Toolkit, který je nástrojem sloužícím k získávání dat z databáze Wikipedie.

V šesté kapitole je popsán a zobrazen návrh, který bude sloužit čtenáři k základní představě struktury výsledného vyhledávacího systému. Dále jsou zde popsány popisy a rozbory některých kroků tohoto návrhu. Nakonec kapitoly je rozebrán způsob výpočtu klíčových slov jednotlivých článků.

Sedmá kapitola je v první fázi věnována postupu nutnému ke zprovoznění samotného programu. Konkrétně jsou zde popsány úpravy databázového systému PostgreSQL, instalace mediawiki a databáze. V neposlední řadě je zde rozebrán postup pro zprovoznění Wikipedia Miner Toolkitu a samotného vyhledávacího programu. Dále se tato kapitola podrobně zabývá jednotlivými kroky implementace popsaných v návrhu. Také jsou zde popsány případné problémy a komplikace v dílčích fázích a jejich následné řešení.

V předposlední osmé kapitole je prováděno testování vyhledávacího algoritmu, kde jsou prováděny tři kategorie testů.

Závěrečná devátá kapitola obsahuje závěr, který shrnuje a hodnotí celou práci a také uvádí možné rozšíření a budoucí vývoj.

Kapitola 2

Vyhledávání informací

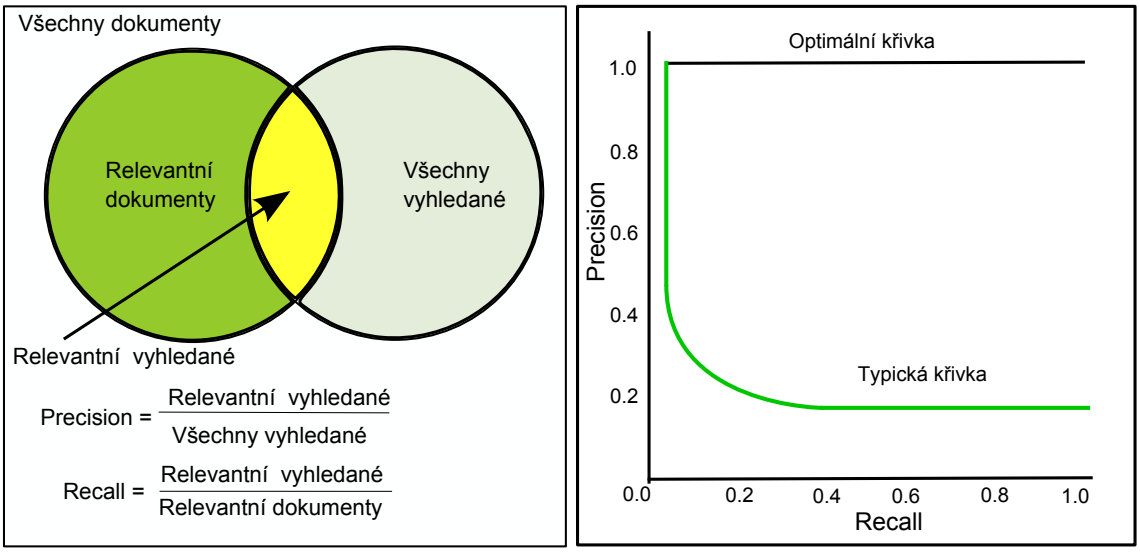
Pro vyhledávání informací (dále jako IR) je potřeba mít samotné informace, jak již napovídá název. Co je informace? Podle business dictionary [30] to jsou data, která již byla ověřená, přesně definována a jsou aktuální. Je potřeba, aby informace byly specifické a organizované pro požadovaný účel. Samotná informace tak potom zvyšuje uspořádanost systému. Narůstání entropie (míra neurčitosti náhodného procesu) tedy značí přechod do méně uspořádaného stavu. Pro vyhledávání informací je tedy důležité mít vhodné data.

2.1 Rozvoj vyhledávání informací

Podle [27] si americká armáda v roce 1940 jako první uvědomila, že je potřeba vyřešit problém ukládání a rychlého opětovného vyhledání dat. Důvod je zde totiž ten, že potřebovali rychle vyhledávat v datech, které získali špionáží od Němců. Jako první pak o tomto problému veřejně promluvil Vannevar Bush ve svém článku "As we may think"v roce 1945. Následně byly v 50. a 60. letech 20. století představovány experty různé vyhledávací systémy. Do počátku 70. let pak byly postupně představeny takové metody, které již kvalitně fungovaly na malých vzorcích dat (například Cranfield kolekce obsahující několik tisíc dokumentu). Systémy, které byly schopny vyhledávat ve větších vzorcích dat, pak byly vytvořeny a představeny v 70. letech. V roce 1992 následně Ministerstvo obrany USA spolu s Národním Institutem pro Standardy a Technologie (NIST) spolufinancuje Konferenci zaměřenou na vyhledávání informací (TREC). Zde bylo cílem vytvoření infrastruktury, která je potřebná pro vyhledledávání informací ve velkých kolekcích dat. Podle [4] se pak v posledních dvaceti letech oblast vyhledávání informací (dále jen IR) podstatně rozrostla. Ještě nedávno nebyl o oblast IR tak velký zájem a byla to spíše oblast pro knihovníky a informační experty. Změna nastala s nástupem World Wide Web neboli WWW. Jedná se o interaktivní médium, které umožňuje levný a jednoduchý přístup. Ten se rozrostl do nečekaných měřítek. Skutečnost, že Web se stal přístupný všem a lidé mohou zveřejňovat co si zamanou, způsobuje, že vyhledávání informací se stává ještě těžším. Proto se dnes podle [4] v IR vyvíjejí oblasti jako jsou modelování, klasifikace dokumentů, uživatelské rozhraní, systémová architektůra, filtrování a mnoho dalších.

2.2 Vyhledávání informací a vyhledávání dat

Podle profesorů Ricardo Baeza-Yatese a Berthiera Ribeiro-Neta [4] je **vyhledávání informací** (dále jen IR) část počitačové vědy, která studuje získávání informací ze sbírky



Obrázek 2.1: Zobrazení parametrů recal a precision

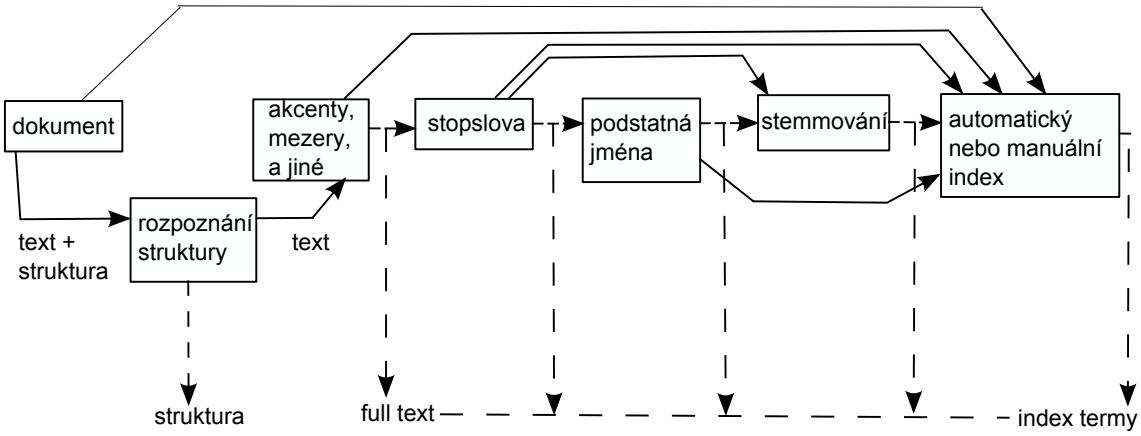
písemných dokumentů (jde o jednotku vyhledávání, která může být kapitolou, článkem, webovou stránkou, či celou knihou). Vyhledané dokumenty pak mají za cíl uspokojit potřeby uživatele (user information need), což znamená, že výsledek musí být použitelný a relevantní.

Definování potřeb uživatele ale není v IR tak jednoduchou záležitostí. Požadavek uživatele je totiž zadán v přirozeném jazyce. Ten obsahuje spoustu nejednoznačností a není moc dobře strukturovatelný. Proto musí být dotaz nejprve zpracován a převeden do takové formy, kterou lze použít pro dotazování. Dalším úkolem IR je, že musí nějakým způsobem interpretovat data a zobrazit výsledky podle relevance. Snahou je zde pak také nezobrazovat dokumenty, které nejsou relevantní. Realita je ale jiná. Mnohdy systémy naleznou mnoho nechtěných dokumentů. Podle autorů Grossman a Frieder [10] máme dvě hodnoty, které nám pomohou určit, jak kvalitně se informace vyhledávají. První je precision neboli přesnost. Ta určuje poměr relevantních vyhledaných dokumentů ke všem vyhledaným dokumentům. Druhá hodnota je pak recall. Ta podle [10] určuje jestli je počet důležitých vzorků dostačující. Například pokud nám systém vrátí 5 vhodných dokumentů, ale ve skutečnosti jich existuje stovky, pak výsledná množina vrácených dokumentů není dobrá. Získání parametru recall ale není triviální. Porovnání obou parametrů lze vidět na obrázku 2.1

Oproti tomu **vyhledávání dat** (dále jen DR) má za cíl vyhledat přesně definovaná data. Chybně nalezený prvek by zde mohl mít fatální následky. K tomuto případu ale u DR nedochází, protože dotaz je definován přesnými požadavky jako je například regulární výraz. V DR se tedy pracuje s dobře definovanou strukturou a sémantikou.

2.3 Reprezentace dokumentů

Z předchozího odstavce tedy víme, že díky WWW máme velké množství dokumentů, ve kterém chceme nějakým způsobem vyhledávat. Je proto důležité zvolit vhodnou reprezentaci jednotlivých dokumentů pro efektivní vyhledávání. Podle [4] máme dva základní typy



Obrázek 2.2: Logický pohled na dokument - od fulltextu až po množinu index termů [4]

reprezentace dokumentů, které poskytují logický pohled na dokument. První z nich je reprezentace pomocí **index termů neboli klíčových slov**. V [4] se tvrdí, že jde o předem vybrané termy, které mohou být použity při vyhledávání požadovaného dokumentu. Tyto slova v podstatě sémanticky pomohou definovat hlavní myšlenku dokumentu. Ve většině případů se jedná o podstatná jména (ta totiž mají nějaký význam samy o sobě). V neposlední řadě je vhodné vyhodnotit prioritu všech index termů. Term, který se totiž bude vyskytovat ve většině prohledávaných dokumentů, pro nás nemá takový význam jako term, který se bude vyskytovat například jen v pěti dokumentech. Hodnotě, která popisuje relevanci daného index termu se podle [4] říká **váha**. Mějme tedy k_i jako index term a d_j jako dokument. Váha se pak značí $w_{i,j} > 0$. Ta je pak asociována s dvojicí (k_i,d_j). Váhy jednotlivých termů jsou pak na sobě nezávislé (znalost $w_{i,j}$ a páru (k_{i+1}, j) neřekne nic o $w_{i+1}, j)$. Samotná hodnota váhy je pak určena podle toho, kolikrát se slovo objevilo v dokumentu. Například index term, který se neobjeví v dokumentu vůbec, má váhu $w_{i,j}$ rovnu 0. Váha tedy určuje důležitost daného index termu pro sémantiku. Získávání těchto klíčových slov z textu, pak může probíhat automatickým vygenerováním a nebo to může být prováděno nějakým člověkem (specialistou).

Druhou reprezentací dokumentů je podle [4] fulltextová reprezentace. Zde je dokument reprezentován celým svým obsahem. Full text je bezesporu nejracionálnějším pohledem na dokument, ale jeho použití přináší velké nároky na výpočetní čas. Vyhledávání v této reprezentaci dokumentů pak tedy potřebuje moderní a výkonné počítače. I přes možnost použití výkonných počítačů je potřeba zredukovat jednotlivé texty na jejich gramatický kořen pro snížení časové náročnosti. Tato metoda se podle [4] nazývá **stemming**. Při této metodě například vyhledávače zredukují skloňované slovo a vytvoří z něj infinitiv. V praxi to vypadá tak, že pokud uživatel zadá do vyhledávače slovo, které nemá tvar infinitivu, ale je skloňované, vyhledávač ve většině případů nabídne jen výsledky zobrazující infinitiv. Podle [4] se zde také provádí odstranění tzv. **stopwords**. Jde o slova, které se v článcích velice často vyskytují a nemají pro nás takovou informační hodnotu jako ostatní slova. Jsou to většinou předložky nebo spojky.

V [4] se lze také dočíst, že existuje několik dalších pokročilých logických pohledů na dokument. Ty mohou být schváleny IR systémem. Jak je zobrazeno na obrázku 2.2 vnímáme, že logická reprezentace dokumentu je souvislým prostorem, ve kterém se můžeme pohybovat od fulltextu směrem k vyšším úrovním specifikace. Ty už jsou specifikovány člověkem,

který se na danou oblast specializuje.

Indexy

Dalším vylepšením a zrychlením vyhledávání jsou indexy. Podle [4] se jedná o datovou strukturu založenou na textu a umožňuje urychlení vyhledávání. Jsou v podstatě základem každého moderního vyhledávacího systému. Vyhledávač by musel skenovat každý dokument, což bezesporu vyžaduje velké množství času. Pokud bychom měli například 10 000 dokumentů, tak vyhledávání za použití indexů by trvalo několik milisekund oproti několika hodinám u sekvenčního vyhledávání (v případě velkých dokumentů). Indexy se tak nejčastěji používají u starších a často vyhledávaných dat.

Kapitola 3

Modely vyhledávání informací

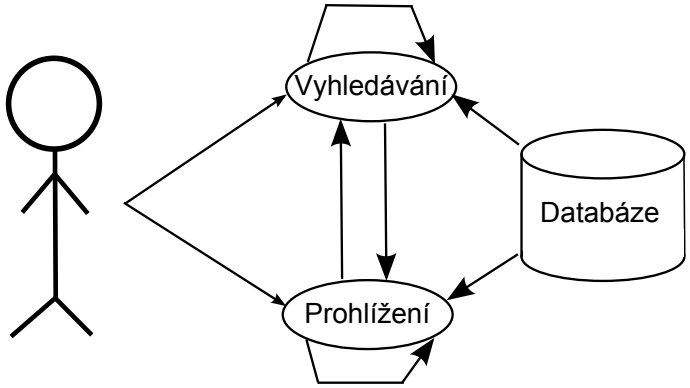
3.1 Základní rozdělení

Dle [4] existuje základní rozdělení uživatelského dotazu na retrieval (vyhledávání) a browsing (prohlížení). Retrieval je takové vyhledávání, kde uživatel ví přesně co chce (například vyhledání adresy konkrétní školy). Naopak browsing je prohlížení výsledných dat, ale uživatel zde nehledá nějaký konkrétní výsledek (například uživatel zadá cestování, kde najde dokumenty o cestování do Francie. Následně tak zadá dotaz na vyhledávání o Francii a tak to může pokračovat). To lze také vidět na obrázku 3.1.

Pro účel této práce se nyní zaměřím na retrieval tedy vyhledávání. Podle [4, 10] máme dva základní operační módy vyhledávání a to jsou Ad hoc a filtrování.

Ad hoc je takový mód ve kterém jsou dokumenty statické nebo relativně statické. Jinými slovy množina dokumentů se nemění. Ty jsou indexovány podle priority k uživatelskému dotazu. Dotaz je pak zobrazen a všechny dokumenty, která se zdají být relevantní jsou seřazeny podle spočítané podobnosti.

Filtrování je proti tomu opakem. Zde jsou statické dotazy, ale mění se množina dokumentů. Speciální variantou u filtrování je routing. Zde se ještě navíc oproti filtrování provádí seřazení vzorků podle toho, jak jsou pro uživatele zajímavé. Konečnému uživateli tak stačí procházet první vzorky. Samotné filtrování má ale ve skutečnosti také vnitřní řazení. To je ale založeno na tzv. Tresholdu (zde má význam práhová hodnota), kde systém zobrazí daný dokument, pokud je hodnota větší než Treshold. V opačném případě dokument zahodí.



Obrázek 3.1: Zobrazení použití typů uživatelských dotazů

Samotné vyhledávání se podle [4] dělí na Klasické modely a Strukturované modely. Nyní se zaměřím na klasické modely do kterých spadají tři základní a to boolovský model, vektorový model a pravděpodobnostní model. Ty se tedy nyní pokusím detailněji rozebrat.

3.2 Booleovský model

První modelem je podle [4] booleovský model, který je velice jednoduchým modelem. Je založen na **teorii množin a booleovské algebře**.Jelikož je koncept množin dostatečně jednoduchý pro představivost, tak model poskytuje principy, které jsou velice snadno pochopitelné pro běžného uživatele systému informačního vyhledávání. Dotazy jsou zde pak specifikovány jako booleovský výraz, jenž poskytuje pevně určenou sémantiku. Díky své jednoduchosti tak model získal v minulosti velkou pozornost na poli vyhledávání a byl použit v mnoha komerčních bibliografických systémech.

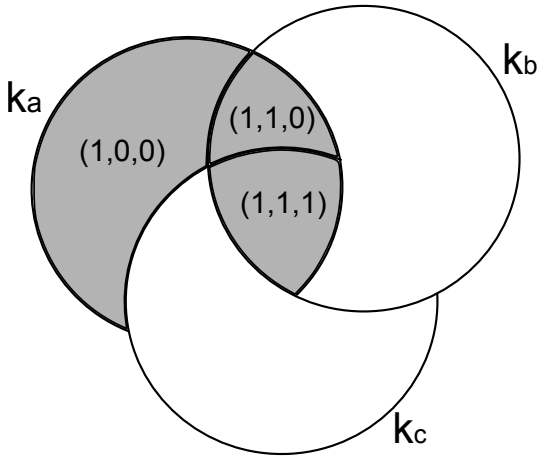
Na druhou stranu má booleovský model ale také mnoho podstatných nevýhod. Jednou z nich je skutečnost, že booleovský model je založen jen na dvouúrovňovém rozhodovacím systému. Jinými slovy výsledek se zde dělí na relevantní a irelevantní dokumenty, což snižuje kvalitu výsledku vyhledávání. Neexistuje zde totiž žádné měřítko, podle kterého by se jednotlivé vyhledané dokumenty třídily vzestupně, či sestupně podle relevantnosti. Právě z tohoto důvodu se booleovský model používá spíše na datové vyhledávání (a ne na vyhledávání informací). Další velkou nevýhodou je skutečnost, že booleovské výrazy mají přesnou sémantiku. To způsobuje, že není vůbec jednoduché transformovat informační požadavek na výrazy booleovy algebry. Tím se samozřejmě ztíží práce uživatelů, kteří musí mnohdy složitě transformovat své dotazy na booleovské výrazy. Přestože má model tyto podstatné nevýhody, podle autorů [4] je stále dominantním modelem na poli komerčních dokumentových databází a stává se základním modelem pro ostatní.

Tento model je jednoduše založen na skutečnosti, že váha každého z index termů zde nabývá jedné ze dvou hodnot 0 nebo 1 ($w_{i,j} \in \{0,1\}$). Samotný dotaz zapsaný pomocí booleovy algebry se skládá z index termů, jenž jsou spojeny pomocí třech spojek a to: not (negace), and (konjunkce) a or (disjunkce). Z tohoto důvodu jednotlivé dotazy můžeme reprezentovat disjunktivními nebo konjunktivními vektory. Příkladem zde může být Disjunktní normální forma. Autoři Ricardo Baeza-Yates a Berthier Ribeiro-Neto mají ve své publikaci [4] uveden také jeden příklad, kde dotaz $[q = k_a \wedge (k_b \vee \neg k_c)]$ lze zapsat v disjunktní normální formě jako $[\vec{q}_{dnf} = (1,1,1) \vee (1,1,0) \vee (1,0,0)]$. Zde pak každá složka představuje binární váhový vektor, který je spojen s trojicí (k_a, k_b, k_c). Tyto vektory se pak nazývají konjunktivní komponenty vektoru $\vec{q}_{dnf}$. Výše popsaný příklad lze pro lepší představivost vidět na obrázku 3.2.

Podobnost dokumentu d_j a dotazu q je podle [4] definována takto :

$$sim(d_j, q) = \begin{cases} 1, & pokud\ \exists \vec{q}_{cc} | (\vec{q}_c c \in \vec{q}_d nf) \wedge (\forall k_i, g_i(\vec{d_j}) = g_i(\vec{q}_c c)) \\ 0, & jinak \end{cases} \tag{3.1}$$

V tomto vzorci platí: $\vec{q}_{cc}$ je libovolný prvek od $\vec{q}_{dnf}$, což je disjunktivní normální forma dotazu q. Vzorec tedy říká, že jestliže platí $sim(d_j, q) = 1$, pak dokument d_j je relevantní vůči dotazu q. V opačném případě model tvrdí, že dokument není relevantní.



Obrázek 3.2: Zobrazení třech komponent pro dotaz $[q = k_a \wedge (k_b \vee \neg k_c)]$ [4]

Rozšíření

Podle [10] lze provést rozšíření booleovského modelu. Základní myšlenka je, že se váha každého termu zakomponuje do každého termu v dotazu a v dokumentu. Tedy místo jednoduchého hledání množiny index termů jsou váhy termů začleněny do ohodnocení dokumentu. Mějme například dotaz t_1 OR t_2, který odpovídá dokumentu obsahující t_1 s váhou w_1 a t_2 s váhou w_2. Pokud jsou obě váhy w_1 a w_2 rovny 1, pak dokument, ve kterém se vyskytují, dostane nejvyšší ohodnocení. Naopak pokud dokument neobsahuje ani jeden z těchto termů t_1 a t_2, pak daný dokument dostane nejnižší ohodnocení. Samotná míra relevance je spočtena jako Eklidovská vzdálenost z bodu (w_1, w_2) do počátku. Pro samotný výpočet podobnost dokumentů (ve vzorcích jako SC), jenž obsahuje termy t_1 a t_2 s váhami w_1 a w_2, se použije vzorec:

$$SC(Q, d_i) = \sqrt{(w_1)^2 + (w_2)^2} \tag{3.2}$$

Pokud bychom měli váhy 0,5 u w_1 a u w_2, pak by koeficient vypadal takto:

$$SC(Q, d_i) = \sqrt{(0.5)^2 + (0.5)^2} = 0.707 \tag{3.3}$$

Nejvyší hodnota koeficientu je, jak již bylo řečeno, v případě že w_1 a w_2 jsou rovny 1. V tomto případě je hodnota koeficientu rovna 1.414. V případě, že chceme, aby byl výsledek v rozmezí 0 a 1, pak je přidána normalizace $\sqrt{2}$. Samotný výpočet koeficientu pak vypadá takto:

$$SC(Q_{t_1 \vee t_2}, d_i) = 1 - \frac{\sqrt{(w_1)^2 + (w_2)^2}}{\sqrt{2}} \tag{3.4}$$

Tento výpočet koeficientu předpokládá, že pracujeme s dotazem, který obsahuje booleovský OR $(t_1 \wedge t_2)$. Lze ale jednoduše provést rozšíření, které bude brát v potaz výpočet pro booleovský AND. Místo výpočtu vzdálenosti k počátku se bude počítat vzdálenost k bodu (1,1). Výsledný vzorec pak vypadá takto:

$$SC(Q_{t_1 \wedge t_2}, d_i) = 1 - \frac{\sqrt{(1 - w_1)^2 + (1 - w_2)^2}}{\sqrt{2}} \tag{3.5}$$

Shrnutí

U boolovského modelu nemůže nastat částečná shoda. Buď je dokument relevantní nebo je irelevantní. To můžeme vidět také na příkladu dokumentu $\vec{d_j} = (0, 1, 0)$, který i když obsahuje index term k_b, tak i přesto je nerelevantní k dotazu $[q = k_a \wedge (k_b \vee \neg k_c)]$. Tato vlastnost se může jevit jako velká nevýhoda, protože vyhledání může vést k nalezení příliš málo dokumentů nebo naopak příliš mnoho. Hlavní výhodou jsou zase bezesporu jeho jednoduchost a čistý formalismus (čerpáno z [4]).

3.3 Vektorový model

Podle [4] samotný booleovský model je jako základ velice užitečný, ale i přesto je třeba si uvědomit, že použití binárních vah je velice neefektivní a slabé. Dalším modelem, který se toto bude snažit vyřešit je vektorový model. Je potřeba nalézt takovou metodologii, ve které bude možné použít i částečnou shodu. To je dosaženo použitím takových hodnot vah jednotlivých index termů dotazů a dokumentů, které nejsou binární. Tyto váhy jsou pak použity k výpočtu **míry podobnosti (degree of similarity)** dokumentu v systému a uživatelského dotazu. Po vyhodnocení všech podobností u jednotlivých dokumentů se pak provede sestupné seřazení podle vypočtené míry podobnosti. Výsledek tedy bude vypadat tak, že nejvíce relevantní dokumenty jsou na začátku výsledné množiny. Uživateli následně stačí prohledat prvních pár dokumentů, kde pozná, jestli nalezl co chtěl. Výsledným efektem je tedy větší přesnost výsledné množiny (ve smyslu, jak dokáže výsledek uspokojit potřebu uživatele) oproti booleovskému modelu.

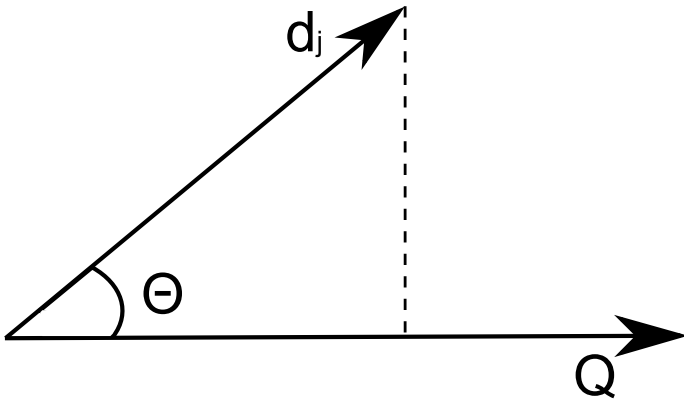
Jak již bylo řečeno ve Vektorovém modelu se stejně jako v booleovském pracuje s váhou $w_{i,j}$, která souvisí s dvojicí (k_i, d_j). Rozdíl je zde tedy ten, že hodnota váhy je kladná a nebinární. Dále se zde oproti booleovskému modelu pracuje s vektorovou reprezentací uživatelského dotazu q a dokumentu d_j (viz [4]). To lze vidět na obrázku 3.3.

V tomto modelu je míra podobnosti dokumentu a dotazu určena jako korelace mezi vektory $\vec{q}$ a $\vec{d_j}$. Tato korelace pak může být například vypočtena jako kosínus úhlu mezi těmito dvěma vektory. Vzorec pro samotný výpočet podobnosti (dále také jako sim) podle [4] vypadá takto:

$$sim(d_j, q) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}} \tag{3.6}$$

Zde faktor $|\vec{q}|$ neovlivní samotné seřazení dokumentů, protože je dotaz pro všechny dokumenty stejný. Oproti tomu $|\vec{d_j}|$ se mění a tak provádí normalizaci v prostoru všech dokumentů.

Jelikož ve Vektorovém modelu mohou již hodnoty vah nabývat hodnot větších a rovno 0, pak samotná hodnota sim může být v intervalu $< 0, 1 >$. Díky tomu se nemusí uvažovat, které z dokumentů jsou relevantní, ale místo toho podle [4] můžeme seřadit všechny dokumenty dle jejich míry podobnosti (degree of similarity). Je zde ale také potřeba určit, které dokumenty už nejsou významné pro daný dotaz. K tomu slouží určení tzv. **tresholdu**, což je hodnota, která určuje hranicí míry podobnosti, jenž je ještě přípustná. Pro samotné spočítání ohodnocení se ale nejdříve musí určit, jakým způsobem se vyhodnotí váhy jednotlivých index termů.



Obrázek 3.3: Zobrazení porovnání kosínu úhlu mezi vektorem dokumentu d_j a dotazu q. Převzato z [4]

Váhy jednotlivých index termů mohou být spočteny mnoha způsoby. Autoři Baeza-Yates a Ribeiro-Neto [4] se zde odkazují na práci panů Saltona a McGilla [26], kteří ve své publikaci popisují různé techniky na výpočet vah index termů. V této práci jsem se zaměřil jen na jednu z nich, která je považována podle autorů [4, 10] za nejefektivnější a je založena na zakladních principech teorie shlukování. Jde o metodu váhování zvanou TF-IDF (skládá se ze dvou druhu váhování a to TF a IDF). Nejdříve bych tedy provedl menší teoretický úvod do teorie shlukování.

3.3.1 Teorie shlukování

Mějme kolekci objektů C a *nejednoznačný popis množiny* A, kde cílem jednoduchého shlukovacího algoritmu je rozdělit kolekci C na dvě množiny. První je množina, která se skládá z prvků, které jsou v relaci s množinou A a druhá která se skládá z prvků, které nejsou v relaci s množinou A. Nejednoznačný popis množiny zde znamená, že nemáme kompletní informace pro rozhodnutí, které prvky jsou v dané množině A a které nejsou. Například pokud chce uživatel vyhledat, která auta mají porovnatelnou cenu s automobilem určité značky, tak není přesně definováno co slovo porovnatelnou znamená - neexistuje zde přesný popis množiny A. Více propracované algoritmy pak mohou být schopny rozdělit prvky do více tříd. Například pacienti doktora specializovaného na rakovinu se mohou dělit na více skupin (závěrečné stádium, pokročilé stádium, metastáze, diagnostikován a zdravý). Zde také samotné popisy tříd mnou být nejednoznačné a problémem je proto také rozhodnutí do které ze skupin patří nový pacient. V naší problematice nás naštěstí zajímá jen jednoduchá verze shlukovacího algoritmu (taková která zohledňuje rozdělení jen do dvou tříd), protože nám stačí vědět pro každý z dokumentů v systému, zda je relevantní vzhledem k dotazu nebo ne (čerpáno z [4]).

Uvažujme nyní, že dokumenty budou kolekce C a uživatelským dotazem bude již zmíněný nejednoznačný popis množiny A. Zde celý IR problém může být zredukován na takový problém, kde rozhodujeme, který prvek je v množině A a který není. V teorii shlukování musíme vyřešit problém, který má podle [4] dvě části.

1. Je potřeba určit vlastnosti prvků, které nejlépe popisují ty prvky, které se nachází v množině A. Tato množina vlastností určuje tzv. vnitro-skupinovou podobnost (*intra-cluster similarity*).

2. Je třeba také určit takové vlastnosti prvků, které umožňují rozpoznat mezi-skupinovou rozdílnost (*inter-cluster dissimilarity*).

Hlavní snahou nejlepších algoritmů je vyvážení těchto jevů. Ve Vektorovém modelu by se dala vnitro-skupinová rozdílnost vyjádřit, jako četnost termů k_i v dokumentu d_j. Tato četnost bývá často označována jako **tf faktor** (*term frequency*) a umožňuje určit, jak dobře daný term popisuje obsah dokumentu. Jedinou nevýhodou tf faktoru je podle [5], že neřeší délku dokumentu. Mezi-skupinová rozdílnost je ve vektorovém modelu určena jako frekvence termu k_i mezi jednotlivými dokumenty v kolekci. Tento faktor je obvykle označován jako frekvence inverzního dokumentu (*inverse document frequency*) neboli **idf faktor**. Důvodem použití tohoto faktoru je snaha rozlišit, které termy se často vyskytují v mnoha dokumentech. Při častém výskytu totiž daný term nemá pro určení relevantnosti dokumentu velký význam.

Pokud chceme vypočíst váhy jednotlivých dokumentů musíme pro ně nejdříve vyčíslit oba parametry váhování tf a idf. Autoři [4] uvádí pro TF faktor vzorec:

$$f_{i,j} = \frac{freq_{i,j}}{max_l freq_{l,j}} \tag{3.7}$$

kde $freq_{i,j}$ je četnost index termu k_i v dokumentu d_j a $max_l freql, j$ je spočteno jako výskyt nějakého termu k_l v dokumentu d_j, ve kterém se vyskytuje nejčastěji. Autoři Sulton a Buckley navrhují výpočet tf faktoru jako:

$$f_{i,j} = 0.5 + \frac{0.5 \cdot freq_{i,j}}{max_l freq_{l,j}} \tag{3.8}$$

Vzorec pro IDF faktor pak podle [4, 10] vypadá takto:

$$idf_i = log\frac{N}{n_i} \tag{3.9}$$

N je celkový počet dokumentů a n_i je počet dokumentů, které obsahují term k_i.

Podle [10, 4] výsledný vzorec pro tf-idf váhování je:

$$w_{i,j} = f_{i,j} \times idf_i \tag{3.10}$$

kde po dosazení

$$w_{i,j} = 0.5 + \frac{0.5 \cdot freq_{i,j}}{max_l freq_{l,j}} \times log\frac{N}{n_i} \tag{3.11}$$

Shrnutí

Vektorový model má tedy 3 základní výhody:

- Použitím výpočtu vah jednotlivých termů se vylepšila schopnost vyhledávání.
- Schopnost nalézt částečnou shodu, umožnila vyhledat i ty dokumenty, které mají s dotazem jen částečnou shodu.
- Konečné kosínus ohodnocení, které umožňuje seřadit jednotlivé dokumenty podle jejich míry podobnosti vzhledem k dotazu.

Za nevýhodu vektorového modelu lze pak považovat skutečnost, že jednotlivé index termy jsou vzájemně nezávislé. V praxi podle autorů Baeza-Yatese a Ribeiro-Neta [4] se ukázalo, že nezávislost index termů není zas takový problém. Ačkoliv jde o jednoduchý model, je velice schopný a odolný při vyhledávání a řazení dokumentu podle relevantnosti. Vytváří takovou množinu výsledků, kterou lze velice těžko vylepšit bez nějakého rozšíření dotazu nebo zpětné vazby. Existuje velké množství alternativních metod, které byly porovnávány s vektorovým modelem, ale jejich výsledky se jeví jako porovnatelné a nebo horší. V neposlední řadě jde o velice jednoduchý a rychlý model. Z těchto důvodů se tento model vyhledávání stal populárním (čerpáno z [4]).

3.4 Pravděpodobnostní model

Dalším modelem ze skupiny klasických modelů je pravděpodobnostní model. Budu se snažit rozvést takový klasický pravděpodobnostní model, který popisují pánové Roberston a Sparck Jones ve své publikaci [25]. Tento model je také znám pod názvem binární nezávislé vyhledávání (*binary independence retrieval (BIR)*). Mým hlavním cílem nebude detailně rozebrat tento model, ale popsat klíčové funkce tohoto modelu.

Podle [4] se pravděpodobnostní model snaží zachytit IR problém uvnitř pravděpodobnostního rámce. Máme-li uživatelský dotaz, pak je zde také množina obsahující jen relevantní dokumenty a žádné jiné. Tuto množinu lze také označit jako ideální množinu. Pokud máme popis takovéto ideální množiny, pak pro nás není problémem vyhledání odpovídajících dokumentů. Proto je potřeba vymyslet nějaký dotazovací proces, pomocí kterého určíme vlastnosti ideální množiny (obdobně jako u interpretace IR problému shlukování). Problémem je zde totiž skutečnost, že nevíme jaké tyto vlastnosti jsou. Vše co v podstatě víme je, že zde jsou index termy, jejichž sémantika by měla pomoci určit charakteristiku těchto vlastností. Jelikož tyto vlastnosti neznáme v době dotazu, je potřeba vynaložit úsilí pro odhad, o jaké vlastnosti by mohlo jít. Jinými slovy je potřeba vytvořit inicializační odhad. Ten nám umožní vytvořit předběžný pravděpodobnostní popis ideální množiny, jenž je použita k vyhledání první množiny dokumentů. Následně je důležitá uživatelská interakce, jejíž účelem je vylepšení pravděpodobnostního popisu ideální množiny.

Samotná uživatelská interakce by mohla vypadat takto: Uživatel si prohlédne vyhledané dokumenty a vybere ty, které jsou relevantní a které ne (probíhá to tak, že prohlédne jen prvních pár dokumentů). Systém následně tyto informace použije ke zdokonalení popisu ideální množiny. Předpokládá se, že opakováním toho procesu se daný popis bude vyvíjet a čím dál více se bude blížit skutečnému popisu ideální množiny, jak tvrdí v [4]. Problémem u tohoto postupu je skutečnost, že vždy musíme hádat počáteční popis ideální množiny.

Pravděpodobnostní model je založen na následujícím předpokladu: Mějme uživatelský dotaz q a dokument d_j v dané kolekci. Pravděpodobnostní model se zde snaží vyhodnotit pravděpodobnost toho, že uživatel bude dokument považovat za relevantní. Model dále pak předpokládá, že pravděpodobnost relevance závisí pouze na dotazu a na reprezentaci dokumentu. Dále také model předpokládá, že existuje taková podmnožina všech dokumentů, kterou uživatel preferuje jako výslednou množinu pro dotaz q. Tato ideální množina, která je označována jako R maximalizuje celkovou pravděpodobnost relevance. Dokumenty, které jsou v této množině R jsou označovány jako relevantní pro daný dotaz. Naopak ty, které se v této množině nenachází, jsou nedůležité pro daný dotaz. čerpáno z [4].

Dále mějme dotaz q, u kterého podle [4] pravděpodobnostní model přiřazuje každému dokumentu d_j jeho míru podobnosti s tímto dotazem q. Tato míra se vyčíslí jako poměr P(d_j

relevantní k q)/P(d_j není relevantní k q) a počítá šanci, že daný dokument bude relevantní vůči dotazu q. Tato šance podle [8, 29] sníží pravděpodobnost chybného rozhodnutí.

U pravděpodobnostního modelu mají index termy binární váhy jako u booleovského modelu, tedy platí $w_{i,j} \in \{0,1\}$, $w_{i,q} \in \{0,1\}$ a dotaz q je podmnožinou index termů. Jak již bylo řečeno výše, množina R obsahuje dokumenty, které jsou relevantní. Oproti tomu $\bar{R}$ je komplementem množiny R. Tato množina tedy obsahuje dokumenty, které nejsou relevantní. Z tohoto pak vycházíme, že $P(R \mid \vec{d_j})$ je pravděpodobnost, že dokument d_j je relevantní k dotazu q a $P(\bar{R} \mid \vec{d_j})$ je pravděpodobnost, že dokument d_j není relevantní k dotazu q. Podobnost dokumentu d_j k dotazu q je pak podle [4] definována jako:

$$sim(d_j, q) = \frac{P(R \mid \vec{d_j})}{P(\bar{R} \mid \vec{d_j})} \tag{3.12}$$

Při použití Bayesova pravidla by byla podle [4] podobnost definována takto:

$$sim(d_j, q) = \frac{P(\vec{d_j} \mid R) \times P(R)}{P(\vec{d_j} \mid \bar{R}) \times P(\bar{R})} \tag{3.13}$$

Zde $P(\vec{d_j} \mid R)$ značí pravděpodobnost náhodného výběru dokumentu z množiny R (relevantní dokumenty). P(R) pak značí pravděpodobnost, že dokument náhodně vybraný z celé kolekce je relevantní. Význam $P(\vec{d_j} \mid \bar{R})$ a $P(\bar{R})$ ve jmenovateli je pak analogicky opačný (u $\bar{R}$ jde o doplňek množiny R). Protože jsou $P(R)$ a $P(\bar{R})$ stejné pro všechny dokumenty v kolekci, můžeme zapsat:

$$sim(d_j, q) \sim \frac{P(\vec{d_j} \mid R)}{P(\vec{d_j} \mid \bar{R})} \tag{3.14}$$

Za předpokladu, že považujeme index termy za nezávislé, můžeme zapsat vztah:

$$sim(d_j, q) \sim \frac{(\Pi_{gi(\vec{d_j})=1} P(k_i \mid R)) \times (\Pi_{gi(\vec{d_j})=0} P(k_i \mid R))}{(\Pi_{gi(\vec{d_j})=1} P(k_i \mid \bar{R})) \times (\Pi_{gi(\vec{d_j})=0}) P(\bar{k_i} \mid \bar{R}))} \tag{3.15}$$

Zde $P(k_i \mid R)$ je pravděpodobnost, že index term k_i je v dokumentu, který je náhodně vybrán z množiny R. $P(\bar{k_i} \mid R)$ je zase naopak pravděpodobnost, že index term k_i není v dokumentu, který byl náhodně vybrán z množiny R. Pravděpodobnosti, které jsou pak ve jmenovateli (pravděpodobnosti s $\bar{R}$) jsou zase významově analogicky opačné jako v předchozím případě. Jelikož víme, že platí $P(k_i \bar{R}) + P(\bar{k_i} \mid R) = 1$, pak s ignorováním konstant, které jsou pro všechny dokumenty stejné, můžeme pomocí logaritmů zapsat:

$$sim(d_j, q) \sim \sum_{i=1}^{t} w_{i,q} \times w_{i,j} \times (log \frac{P(k_i \mid R)}{1 - P(k_i \mid R)} + log \frac{1 - P(k_i \mid \bar{R})}{P(k_i \mid \bar{R})}) \tag{3.16}$$

Tento výraz je pak podle [4] klíčový pro výpočet v pravděpodobnostním modelu.
Jelikož ale na začátku je množina R neznámá, je nutné vymyslet nějakou inicializační metodu pro výpočet $P(k_i \mid R)$ a $P(k_i \mid \bar{R})$. Existuje jich mnoho, ale já zde uvedu jen pár z nich.

Ze začátku nejsou vyhledány žádné dokumenty. Proto se musí provést 2 jednoduché předpoklady a to:

$$P(k_i \mid R) = 0.5 \tag{3.17}$$

$$P(k_i \mid \bar{R}) = \frac{n_i}{N} \tag{3.18}$$

kde n_i je podle [4] počet dokumentu, které obsahují index term k_i a N je zde celkový počet dokumentů v kolekci. Když pak máme takto vyhodnocené počáteční pravděpodobnosti, můžeme provést první vyhledávání dokumentů a jejich následné ohodnocení. Předpokládejme pak, že V je podmnožina takto vyhledaných a ohodnocených dokumentů. Ta může být ustálena například jako r nejvýše ohodnocených dokumentů, kde r je předem určená hranice (threshold). Dále pak máme množinu V_i, která je podmnožinou množiny V a která se skládá jen z těch dokumentů, jenž obsahují index term k_i. Pro vylepšení pravděpodobnostního ohodnocení musíme tedy vylepšit počáteční odhad $P(k_i \mid R)$ a $P(k_i \mid \bar{R})$. To můžeme provést pomocí těchto předpokladů: hodnota $P(k_i \mid R)$ může být vylepšena tak, že se vypočte jako počet dokumentů obsahujících index term k_i k počtu všech vyhledaných dokumentů. $P(k_i \mid \bar{R})$ pak může být vylepšena pomocí přepokladu, že všechny nevyhledané dokumenty nejsou relevantní. Tyto předpoklady lze zapsat zapsat takto:

$$P(k_i \mid R) = \frac{V_i}{V} \tag{3.19}$$

$$P(k_i \mid \bar{R}) = \frac{n_i - V_i}{N - V} \tag{3.20}$$

Tento proces pak může být rekurzivně opakován. Pomocí tohoto postupu jsme schopni vylepšit odhad pravděpodobností bez lidské interakce, kterou naopak vyžadovala původní myšlenka. Poslední dva vzorce špatně fungují s malými hodnotami V a V_i. K vyřešení těchto problému autoři navrhují tyto dva vzorce:

$$P(k_i \mid R) = \frac{V_i + \frac{n_i}{N}}{V + 1} \tag{3.21}$$

$$P(k_i \mid \bar{R}) = \frac{n_i - V_i + \frac{n_i}{N}}{N - V + 1} \tag{3.22}$$

K výpočtu tedy můžeme použít tyto vzorce a nebo můžeme místo toho použít uživatelskou zpětnou vazbu (čerpáno z [4]). Nyní se pokusím shrnout výhody a nevýhody pravděpodobnostního modelu.

Shrnutí

Výhodou tohoto modelu je skutečnost, že dokumenty řadí podle jejich pravděpodobnosti relevance. Na druhou stranu má podle [4] 3 nevýhody. Mezi ně patří nutnost počátečního rozdělení na relevantní a irelevantní. Další nevýhoda je, že není brána v potaz frekvence výskytů index termů (díky tomu že všechny váhy jsou binární). Třetí nevýhodou je pak předpoklad nezavislosti jednotlivých index termů.

3.5 Shrnutí modelů

Nyní se pokusím shrnout výše zmíněné modely a vyberu z nich jeden, který použiji ve své práci.

Ze třech modelů, které jsem popisoval, je určitě nejslabší booleovský model. Jeho hlavní nevýhodou je totiž neschopnost rozpoznat částečnou shodu, což podstatně snižuje jeho

efektivnost a výkonnost. Rozpoznání kvality mezi vektorovým a pravděpodobnostním modelem už není tak jednoduché. Nejprve totiž pan Croft provedl experimenty, které ukázaly, že pravděpodobnostní model vykazuje lepší výsledky a tudíž lepší vyhledávací schopnosti. Později ale pánové Salton a Buckley vyvrátili toto tvrzení díky měřením, které poukázaly na to, že vektorový model vykazuje lepší výsledky na obecných kolekcích dokumentů oproti pravděpodobnostnímu modelu. Tento názor také podle pánů Baeza-Yatese a Ribeiro-Neta zastává většina výzkumných pracovníků, odborníků a lidí z Webové komunity (čerpáno z [4]).

Kapitola 4

Textová data, sémantické slovníky

Nyní už jsme schopni vyhledávat data pomocí modelů vyhledávání informací. Zde byly data vyhledávání informací označovány jako "dokumenty". Jaké jsou tedy typy těchto dokumentů? V této kapitole se je pokusím zkráceně popsat (s výjimkou obrazových dat, které nejsou cílem mé práce). V neposlední řadě zde detailněji popíši Wikipedii.

4.1 Textová data

Jde o základní a nejběžnější formu interpretace dat ve vyhledávání informací (dále jako IR). Je to z toho důvodu, že jde o nejstarší existující formu reprezentace dat. Již cca 5000 let před naším letopočtem totiž byly v oblasti Egypta a Mezopotámie zapisovány informace v textové podobě. Text je také podle [1] přirozený ve všech systémech prakticky stejný a v čase neměnný způsob komunikace. Není proto divu, že z těchto důvodů je pro ně technika vyhledávání nejdokonalejší.

Text lze podle pánů Meadowa, Boyce a spol. [15] považovat za skalární řetězec velké délky. Starší systémy umožňovaly jen pouhé zobrazení daných textů, ale nikoliv vyhledání. Novější systémy oproti tomu považují text za proměnnou, či pole a jsou schopny na základě slov vyhledat texty podobného významu (viz Sémantické slovníky).

Text může být ale podle [15] také považován za pole slov. Zde se potom ignoruje syntaxe a provádí se jen mechanické vyhodnocování počtu výskytů daných slov pole (viz např vektorový model z předešlé kapitoly).

Nejkvalitnější varianta vyhledání v textových datech je ale bezesporu samotné přečtení textů uživatelem, což je ale ve větších textových databázích mnohdy nemožné. Mezi IR systémy, které pracují s textovými daty patří všechny běžné systémy jako jsou knihovní systémy, či webové vyhledávače.

4.2 Sémantické slovníky

Sémantické slovníky (někdy také označovány jako Významové slovníky) jsou takové slovníky, které mají za úkol vysvětlit sémantiku jednotlivých slov. Co je to tedy vlastně sémantika? Podle [28] jde o náuku o významech jednotlivých slov, morfémů a jiných znaků nebo jejich vztahů ke skutečnosti, kterou označují. Toto slovo vzniklo z sémantios neboli znamení, či znak.

V praxi jsou většinou sémantické slovníky seznamy klíčových slov, ke kterým jsou přidruženy krátké popisky jejich významu. Uživatel pak zadá klíčové slovo a systém mu vy-

hledá odpovídající popis významu tohoto slova. Jde tedy většinou o slovníky, jenž obsahují termíny, které jsou odborné nebo v cizím jazyce.

Existuje ještě jeden způsob popisu sémantiky slov, který je v dnešní době populární. Sémantika slova se v něm zapíše pomocí seznamu synonym případně antonym. Uživatel si tak může udělat představu o významu daného slova sám (jen pokud zná sémantiku těchto nabízených synonym a antonym). Tento typ slovníku se nazývá **Thezaurus**.

Slovo Thezaurus je podle [4] řeckého a latinského původu, kde význam je poklad, či pokladnice. Jak již tedy bylo řečeno Thezaurus se skládá z klíčových slov a jim odpovídajících synonym a antonym. Obecně Thezaurus zahrnuje také normalizaci slovníku. Ta pak zahrnuje více složitou strukturu než jsou slova a jednoduchý seznam odpovídajících synonym a antonym. Příkladem může být Thezaurus od Petera Rogata. Ten zahrnuje kromě výše zmíněného také fráze, které vyjadřují význam daných slov. Díky tomu jsou koncepty více komplexní než jen jednoduchá slova.
Podle [4] jsou hlavní cíle Thezauru tyto:

- Poskytovat standardní slovník pro indexaci a hledání.
- Pomoct uživateli s vhodným výběrem termů pro správnou formulaci dotazu
- Poskytovat klasifikovanou hierarchii, která umožní rozšíření a zpřesnění aktuálního uživatelského dotazu podle uživatelských potřeb.

Hlavní myšlenkou je tedy vytvořit řízený slovník, který poskytuje výhody jako jsou normalizace indexování konceptů, redukce šumu, určení index termů, které mají čistou sémantiku a vyhledávání, které je založeno na konceptech a né na slovech.

Hlavními komponentami thezauru jsou jeho 1.index termy, 2.vztahy mezi termy a 3.dispoziční řešení těchto vztahů. *Termy* jsou zde pak indexovací elementy thesauru. Mohou jimi být slova, skupiny slov, či fráze. Většinou jde ale o jednoduchá slova a to podstatná jména. Zpravidla pak platí, že index term v thesauru označuje *koncept*, což je základní sémantická jednotka zprostředkování myšlenek. Tyto koncepty mohou být vyjádřeny skupinou slov, které vzniknou spojením více index termů.

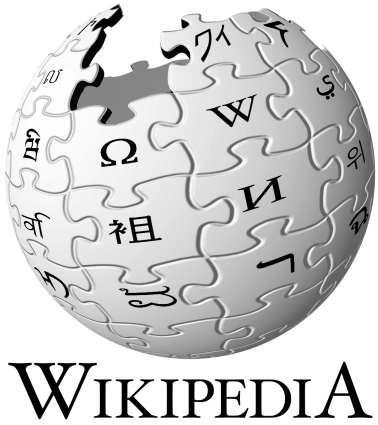
Thesaurus je tedy takový typ sémantických slovníků, které mě budou v této práci zajímat, neboť Wikipedia je obdobou tohoto typu slovníku.

4.3 Encyklopedie (Wikipedie)

Podle [33] slovo Wikipedie vzniklo ze spojení wiki a encyklopedie. Zde je wiki označení takových webů, které umožňují přidávání obsahu (jako u internetových diskuzí) a také jeho editování. Slovo encyklopedie pak představuje strukturované a objemné dílo, které se snaží shrnout lidské vědění a poznání týkajícího se jednoho, či více oborů.

Tato encyklopedie je specifická tím, že na její tvorbě se může podílet kdokoliv z celého světa. Jde totiž o **webovou encyklopedii se svobodným obsahem**, která existuje již ve 250 jazykových verzí různého rozsahu. Jak je již výše zmíněno, Wikipedia je založena na principu wiki. To sebou ovšem nese riziko vandalismu. Každý totiž může vytvařet a editovat libovolné příspěvky (čerpáno z [33]).

Wikipedie je podle [34] jedním z mnoha projektů nadace Wikimedia Foundation Inc. Ta například také spravuje Wikislovník, Wikicitáty nebo Wikiknihy. Cílem této organizace je zajistit další rozvoj a podporu všech projektů. Také je úkolem snaha, aby všechny projekty zůstaly bezplatné.



Obrázek 4.1: Logo Wikipedie

.

Samotná Wikipedie běží podle [16] na **MediaWiki**. To je software, který je také označován jako engine neboli stroj Wikipedie. Program je napsán v PHP za použití databáze MySQL nebo PostgreSQL. MediaWiki je podle [17] navržena pro práci s velkým množstvím uživatelů. Její oblíbenost spočívá hlavně v její výkonnosti a propracovanosti struktury.

4.4 Technologie pro fulltextové vyhledávání

Na poli fulltextového vyhledávání existuje více technologií. Patří mezi ně například Google, Apache Lucene, mnoGoSearch, PostgreSQL Tsearch2 a jiné. Tyto zmíněné technologie nyní popíši.

4.4.1 Google Patent

Google je v dnešní době nejpoužívanější internetový prohlížeč a velkou oblibu má díky své relevantnosti výsledků. Google Patent 20050071741 je pak ve skutečnostni název patentu na Google vyhledávač a jeho hodnocení stránek (Page Rank). Slouží k vyhledávání na webu a proto zohledňuje mnoho ovlivňujících parametrů.

Velký důraz je zde kladen na příchozí odkazy na stránku. Sledovány jsou nejen samotné odkazy, ale také jejich chování, jako jsou změny textu v odkazu, či životnost odkazů. Příklady dalších sledovaných parametrů u **příchozích odkazů** jsou:

- Sledování nárůstu odkazů na stránku. V případě rychlého nárůstu odkazů na stránku může jít o spam, což vyhledávač zohledňuje. Proto je lepší zvyšovat počet odkazů pomalu a konstatně.

- Rozlišování přirozených a nepřirozených odkazů na stránku. Například pokud směřují na stránku odkazy s jiným anchor textem (text fungující jako odkaz na stránku), pak jde o přirozený odkaz. Naopak o nepřirozený odkaz jde v případě většího množství stejných anchor textů na stejnou stránku. Takové nerelevantní odkazy pak Google nezapočítává, protože může jít o spam.

- Pokud anchor text reaguje na změny na stránce na kterou odkazuje, pak bude považován za relevantnější. Google tak chce aktivně reagovat na změny. Například uvedení nového výrobku nebo článku.
- Pokud anchor text bude odkazovat na stránku s jiným tématem, nebude započítán. Ošetřují se tak domény, které změnily vlastníka a obsah.

Tento vyhledávač dále zohledňuje informace získané z registrací domén. Ty které byly registrovány dříve mají vyšší relevanci než ty registrované později. Stejně tak jsou výše hodnoceny ty které byly zaplaceny na delší dobu. Tím se omezí vysoké ohodnocování různých spam domén.

Cílem tohoto vyhledávače je pak také reagovat na sezóní trendy. Například při zadání slova koupaliště, by se v létě měly vracet výše stránky otevřených koupališť naopak v zimě by měly být výše stránky uzavřených koupališť. Dalšími zohledňujícími parametry pak mohou být frekvence aktualizování stránek, či sledování na co uživatel kliká, jak často a jak dlouho je na dané stránce.

Na základě všech výše zmíněných a mnoha dalších parametrů je prováděno vyhledávání a ohodnocování stránek. Samotné google vyhledávání lze použít na vyhledávání ve Wikipedii jen v případě, že je databáze uložená veřejně. V této sekci bylo čerpáno z [24].

4.4.2 Apache Lucene

Lucene je vysoce výkonná, jednoduchá a vyspělá knihovna v jazyce Java, která je vydaná pod open-source licencí Apache. Základní úlohou této knihovny je poskytovat fulltextové vyhledávání a indexační funkce. Oproti Google vyhledávači slouží hlavně k vyhledávání v aplikacích a systémech.
Vlastnosti vyhledávání v Lucene jsou:

- Umožňuje nastavení indexace a následné aktualizace indexů.
- Umí sloučit výsledky a vyřadit duplicity.
- Poskytuje možnost ohodnocení výsledků a jejich seřazení dle relevance pomocí více způsobů: metoda TF-IDF, zvýšení významu jednotlivých polí, či normalizace vah.

V Lucene existuje více typů dotazu pomocí kterých lze provádět vyhledávání:

- TermQuery - na základě klíče
- RangeQuery - na základě textového nebo číselného rozsahu
- BooleanQuery - kombinace dotazů do výrazů (AND, OR, NOT, +, -)
- Jiné - fuzzy vyhledávání, ořezávání pomocí zástupných znaků (*,?), dotazy do polí

Ná závěr bych zmínil, že samotné Lucene je použitelné pro Wikipedii a používá se například pro vyhledávání v anglické wikipedii. V této části bylo čerpáno z [2]a [9].

4.4.3 PostgreSQL Tsearch2

Tsearch2 je modul dostupný pro rozšíření PostgreSQL databáze. Podle [6] jde o nástroj sloužící k fulltextovému vyhledávání v PostgreSQL databázi a umožňuje fulltextové indexování. Tento nástroj pro indexaci textových dat používá speciální indexy, které podstatně urychlují vyhledávání. Dalšími výhodami Tsearch2 jsou:

- Umožňuje rychle a efektivně parsovat jednotlivé textové dotazy.
- Podporuje integraci slovníků a seznamů stop slov, thesaurus slovníky a jiné.
- Existuje zde plná podpora formátu UTF-8.
- Nativně zde neexistuje podpora českého stemování. V Tsearch2 je ale možnost využití generátoru šablon, jenž umožňuje pro jednotlivé slovníky vytvářet podporu Snowball stemmeru.

4.4.4 MnoGoSearch

MnoGoSearch je open source vyhledávač, který je určen na vyhledávání ve webových stránkách, v intranetu nebo v lokálním systému. Je distribuován pod General Public License a skládá se ze dvou částí, kde první je indexovací mechanizmus. Ten prochází html odkazy, o kterých pak ukládá informace do databáze. Druhou částí je webové rozhraní zobrazující výsledky pomocí html formulářů. Hlavní funkce MnoGoSearch jsou:

- Plně podporuje většinu databází (MySQL, PostgreSQL, Oracle, MS SQL a jiné)
- Podpora HTTP, HTTPS, HTTP proxy, NNTP, či FTP
- Existence zabudovaných parserů pro formáty html, xml, plain text nebo dokonce mpeg. Pro ostatní formáty existuje podpora externích parserů.
- Umožňuje řadit výsledky dle relevance, popularity a času modifikace. Je zde také možnost nechat řazení na uživateli.
- Podpora HTDB virtualizace URL.
- Lze provádět více indexací a vyhledávání najednou nad jednou databází.
- Umožňuje indexovat vícejazyčné databáze, kde podporované jazyky jsou detekovány automaticky. Dále také podporuje široký rozsah znakových sad.

V této části bylo čerpáno zde [13].

4.4.5 Shrnutí

Většina výše zmíněných fulltextových technologií pracuje ve třech krocích. Prvním je sběr informací, druhým je zpracování informací do databáze, neboli indexování. Třetím krokem je pak zpřístupnění uživatelům, aby mohli klást dotazy. Kromě výše zmíněných existuje více technologii, které jsou aktuálně použitelné pro MediaWiki k fulltextovému vyhledávání. Příkladem mohou být JODA, Sorl, DBSight, Sphinx a jiné. Pro tuto práci bude ale hlavně z důvodu použití databáze PostgreSQL využit modul Tsearch2.

Kapitola 5

Problematika získávání informací z Wikipedie

V této kapitole se zaměřím podrobněji na encyklopedie, zvláště na Wikipedii, která je stěžejní pro mou práci. Zaměřím se zde na problematiku získávání informací v rámci interaktivně specifikovaných kontextů. Dále zde popíši možné řešení, způsob získávání informací z Wikipedie a také výpočet sémantických podobností jednotlivých částí textu.

5.1 Problematika interaktivně specifikovaných kontextů

V dnešní době je na poli fulltextového vyhledávání zásadní problematika nevhodně a nepřesně definovaných dotazů. Například pokud uživatel zadá v angličtině slovo 'jaguar', u kterého je zamýšleno vyhledávat dokumenty o autech, dotaz uživateli zobrazí mnoho nerelevantních dokumentů o slovu jaguar jako šelmě. V této oblasti je proto v dnešní době hlavním úkolem pomoci uživateli správně definovat dotaz.

Tuto problematiku umožňují řešit interaktivně specifikované kontexty, kde jejich použití podle autoru publikace [7] poskytuje lepší výsledky. Co to znamená a jaký je postup tohoto vyhledávání? Jde o princip, kde po zadání uživatelského dotazu jsou nejdříve vyhledány dokumenty obsahující výraz daného dotazu. Následně je v nalezených dokumentech vybrán text vyskytující se okolo těchto výrazu. Nad těmito texty je následně prováděna extrakce klíčových slov, pomocí kterých je vygenerován a vytvořen nový dotaz. Výběr těchto klíčových slov pak může být ponechán na uživateli. S nově vytvořeným dotazem, který je obohacen o nové slova se provede znovu vyhledávání a opětovné ohodnocení. Autoři publikace [7] jsou přesvědčeni, že tento postup je lepší než původní přístup, kde se nechá výběr dokumentů výhradně na uživatelském dotazu.

Problémem v této oblastni je definice a způsob správného využití kontextů. Podle autorů [7] existuje více možností jejich použití. Příklady mohou být tyto:

1. Prvním z možných řešení a přístupů je takový, kde lze po zadání uživatelského dotazu dát uživateli na výběr z kategorií. Po výběru jedné z nich se provede upravený dotaz, který zobrazí jen dokumenty vybrané oblasti. Například pro již zmíněný dotaz obsahující slovo 'jaguar' by to byly slova 'fauna' a 'cars'.

2. Opakem k předešlému způsobu využití kontextů je pak metodika, kdy jsou automaticky vyhodnoceny všechny dokumenty zobrazené uživateli. Na základě těchto automatických vyhodnocení se provede upravený dotaz. Nevýhodou tohoto přístupu

mohou být rozsáhlé dokumenty, které zabírají mnoho kategorií. Výsledek upraveného dotazu by tak obsahoval velké množství kategorií a nedošlo by k upřesnění dotazu.

3. Dalším ze způsobů možného řešení využití interaktivně specifikovaných kontextů je využití odkazů jednotlivých stránek na jiné. Výpočet klíčových slov se získá z těchto odkazů jednotlivých dokumentů (stránek). Na základě těchto odchozích odkazů se vyhodnotí sémantická příbuznost jednotlivých odkazovaných článků a provede se případné upravení dotazu. Tento princip také použiji ve své práci (viz další podkapitoly).

Každý z těchto zmíněných přístupu má své výhody a také nevýhody a je na tvůrcích vyhledávacího systému, který z nich vyberou.

5.2 Wikipedia Miner Toolkit a model Wikipedie

Jak již bylo řečeno, Wikipedia je objemným repositářem informací. Pro samotné vývojáře jde podle [18] hlavně o velkou a vícejazyčnou databázi konceptů a sémantických vztahů. K výzkumným účelům je zde vytvořen nástroj zvaný Wikipedia Miner Toolkit (dále jako WMT), pomocí kterého lze jednoduše integrovat sémantiku Wikipedie do vlastních aplikací. Současné uložiště například anglické verze obsahuje zhruba 6000 stránek a 20GB sémantických vlastností. WMT proto obsahuje skripty, které jsou schopny bez problémů provést extrakci hierarchie kategorii a odkazů. Tento úkon sice trvá delší dobu, ale poté jsme jednoduše schopni rychle přistupovat ke struktuře. To je díky tomu, že toolkit pracuje s MySQL databází. Data jsou tak naindexována a můžeme k nim okamžitě přistupovat (není tedy potřeba zdlouhavého nahrávání). Jednou z důležitých komponent WMT je Java API, která umožňuje jednoduchý přístup k datům Wikipedie.

Nyní popíši jednotlivé důležité třídy, které používá výše zmíněné API a pomocí kterých lze modelovat strukturu a obsah Wikipedie. Vše o čem budu psát lze vidět na obrázku 5.1, či v publikaci [18].

Page (Stránka)

Všechen obsah Wikipedie je podle [18] prezentován pomocí třídy Page (stránka). Toolkit přiřazuje každé stránce id, název a stručný obsah. Další funkcionalita už pak záleží na tom, o jaký typ stránky tedy jde.

Articles (Článek)

Jde o takovou třídu, která poskytuje většinu obsahu Wikipedie. Každý výskyt této třídy (dále jako článek) popisuje jednoduchý koncept nebo téma. Jejich názvy jsou pak stručné a dobře formulované fráze, jenž mohou být použity jako neklíčová slova v thezauru. Příkladem může být článek o domácím psu, který má název *Pes* a nebo jiný článek o zvířatech, který je zase pojmenován *Domácí mazlíčci*. Z těchto článků je pak možno jednoduše extrahovat první větu, či odstavec, čímž získáme definici daného konceptu.
Ke každému z těchto článků existují odkazy na jiné články, které jsou sémantický tomuto článku blízké. Například výše zmíněné články *Pes* a *Domácí mazlíčci* na sebe mohou mít odkaz. Celkový počet těchto odkazu každého článku pak může poukázat na to, jak je daný článek známý (čerpáno z [18]).

Redirect (Přesměrovač)

Jediným úkolem tohoto typu stránky je podle [18] provést přesměrování na jiný alternativní titul. Například na článek pojmenovaný *Pes* jsou odkazovány tituly *Psi, Feny,* či *Rasy psů.*

Category (Kategorie)

Skoro všechny články jsou organizovány do jedné či více kategorií. Ty jsou strukturované tak, že zde můžeme vyhledávat nadřazené, podřazené články nebo články obdobného významu (na stejné úrovni). K získávání těchto informací slouží třída Kategorie. Příkladem může být kategorie *Psovité šelmy*, kde podkategorie jsou *Pes*, *Šakal* nebo *Vlk* (čerpáno z [18]).

Wikipedia (Wikipedie) a ostatní

Třída Wikipedie je podle [18] jednou z nejdůležitějších v modelu. Její instance je centrálním bodem pro funkcionalitu toolkitu. Kromě toho také umožňuje shromažďovat statistiky o encyklopedii nebo provádí přístup k jednotlivým stránkám pomocí iterace, prohlížení a hledání.

Disambiguations (Rozcestníky)

Další třídou je například ta, která řeší odstranění dvojsmyslu (třída Disambiguations). Tato situace nástává v případě, kdy má více článků stejný název, ale jiný význam. Zde pak třída Disambiguations přidá ještě krátkou frázi, která vysvětlí, jak se od sebe jednotlivé články významově liší. Uživatel si tak může vybrat, o který z článků má zájem.

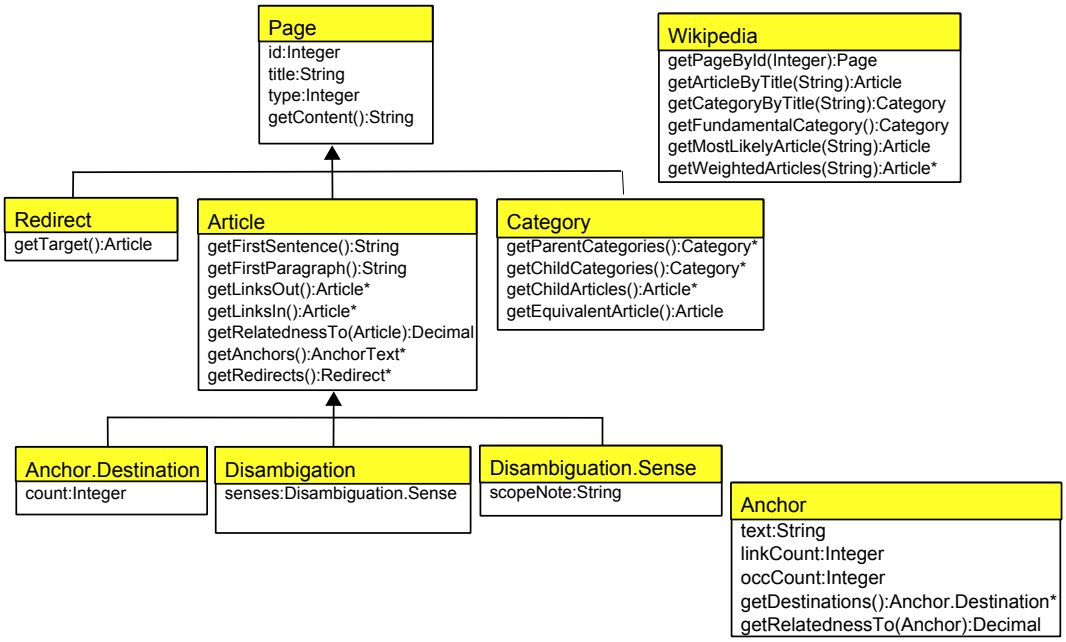
Anchors (Kotvy)

Poslední z významných tříd je třída Anchors. Tato třída umožňuje stejně jako předešlá kódovat mnohoznačnost, ale rozdíl oproti třídě Disambiguations je ten, že odkazy jsou přímé a není zde potřeba provádět zpracování nestrukturovaného textu. Dále také provádí určení, jak je každý smysl pravděpodobný (Například při zadání slova Pes je z 76% pravděpodobné, že jde o odkaz na článek o domácích zvířatech, ze 7%, že jde o odkaz na čínské znamení a méně než 1%, že jde o odkaz na článek o hot dogu). Ostatní třídy nebudu podrobněji rozebírat. Lze je vidět na obr. 5.1 a je možno si o nich více přečíst v publikaci [18] od Davida Milneho.

5.3 Vyhledávání ve Wikipedii

Nejběžnějším způsobem vyhledávání je ten, kdy se vyhledají všechny články, které se mohou týkat vybraného termu. Když tedy například zadám hledat "pes", pak budou vráceny všechny články, které by mohly mít takovýto název. Běžným přístupem je zde pak vyhledání přes názvy stránek a zjištění jejich typů. Podle tohoto postupu jsou pak články (Articles) použity přímo, přesměrování (Redirects) jsou vyřešeny pro vybrané články a následně jsou zjištěny různé významy pomocí Rozcestníků (Disambiguation stránek).

U Disambiguation stránek ale nastává problém jejich automatického zpracování. To je z důvodu, že jsou napsány pomocí čistého textu a také proto, že jednotlivé položky jsou často spíše spojeny s daným názvem než s jeho významem. Na automatické zpracování



Obrázek 5.1: Příklad tříd, vlastností a metod dostupných ve Wikipedii-Miner toolkitu [18]

.

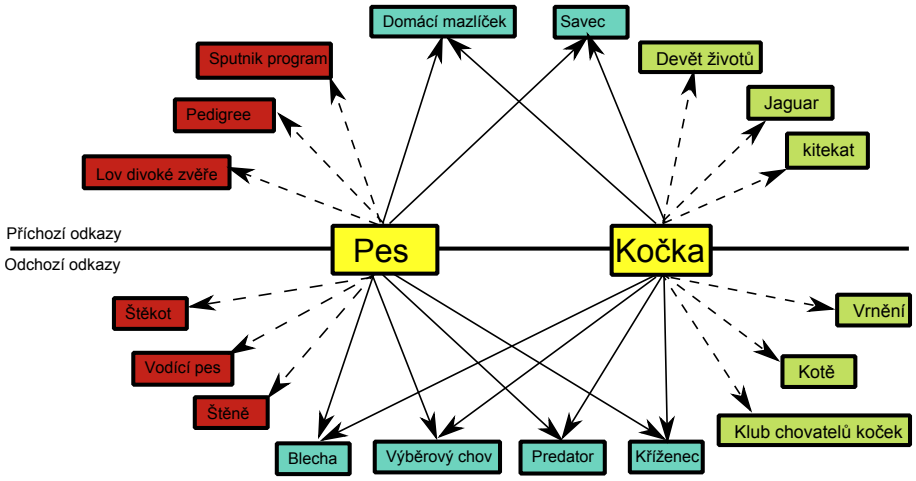
se tedy více hodí již zmíněná třída Anchors, která nepoužívá nestrukturovaný text pro kódování mnohoznačnosti (čerpáno z [18]).

5.4 Výpočet sémantické podobnosti

Jak se dá ale zjistit sémantická podobnost jednotlivých článků? Wikipedia Miner Toolkit (dále jako WMT) podle [18] obsahuje algoritmy pro výpočet sémantické podobnosti, jenž určí, jak jsou různé slova a koncepty navzájem příbuzné. Pomocí WMT lze pak určit například to, že *pes* je na 100% příbuzný s *psovitými šelmami*, ale jen na 19% se *zvířaty*.

Tato sémantická podobnost se generuje pomocí odkazů mezi jednotlivými články wikipedie. Je zde ale nutné vhodně oddělit relevatní od irelevantních odkazů. Například zmíněný článek *Pes* obsahuje spoustu odkazů, protože jde o široký pojem. Jsou zde relevantní odkazy jako *savci, domácí mazlíčci* nebo *výchova psů*. Na druhou stranu je zde také mnoho irelevantních odkazů, jako jsou *palec* nebo *barva*. Proto se musí vždy ověřit, zda je odkazovaný článek skutečně relevantní.

Samotný výpočet podle [18] probíhá tak, že se provede porovnání, kolik mají 2 různé články společných odkazů. Příkladem může být opět článek *Pes* a *Kočka*. Na obrázku 5.2 můžeme vidět, že spolu mají mnoho společných odkazů (v obrázku nejsou zobrazeny všechny pro přehlednost). Příkladem mohou být *blecha, chov, predátor, či kříženec*. Z toho důvodu můžeme říci, že jsou si sémanticky blízké. Na druhou stranu mají také podstatné množství odkazů, které se liší (například *štěně, kotě, vrnění, či štěkot*). Proto můžeme říci, že tyto dva články sémanticky nejsou totožné (podle [18] je jejich vypočtená podobnost 77%).



Obrázek 5.2: Zjišťování sémantické podobnosti dvou článků (čerpáno z [18])

5.5 Vyhodnocení sémantických podobností v článku

Podle [18] Wikipedie vždy zná nějaké informace o článku a je schopna přidat dodatečné informace. Jak lze vidět na obrázku 5.3, lze ze samotného článku pomocí výpočtu sémantické podobnosti (viz předešlý odstavec) určit jednotlivé skupiny souvisejících odkazů (témata). To pak samotnému uživateli usnadní výběr požadovaných informací.

Zjištěná témata také zjednoduší pochopení dokumentu (článku) automatickými systémy. Ty reprezentují daný dokument a koncepty v něm jako "pytle slov". Wikipedie je pak schopná přiřadit k jednotlivým konceptům jejich význam. Před přiřazením významu se ale nejdříve provede detekce relací mezi jednotlivými nalezenými koncepty (viz pravá strana obrázku 5.3, kde můžeme vidět, že byly nalezeny 3 okruhy témat a to: Pes, prezident a technotogie). Zabrání se tak degradaci vyhledávání (mohlo by totiž dojít k přiřazení jiného významu slova, než jaký je zamýšlen v článku).

Postup

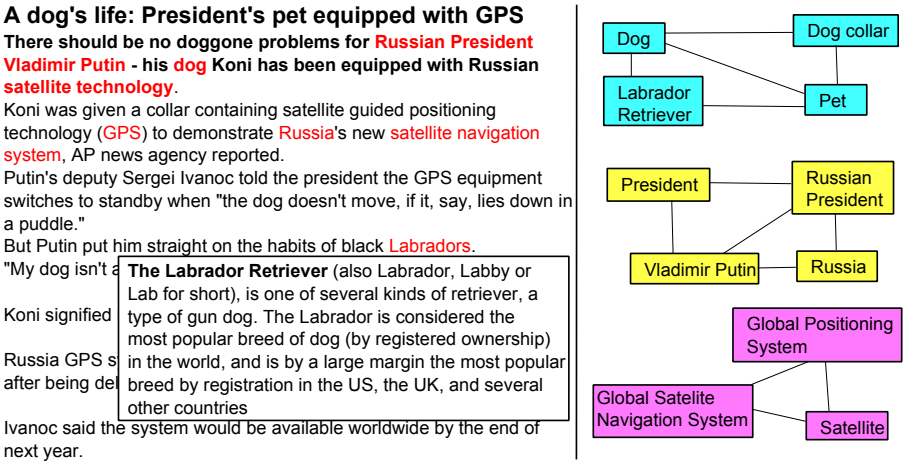
Co tedy již víme? Kromě toho, že jsme schopni vyřešit synonyma objevující se v dokumentu (*GPS* a *satellite guided positioning technology* jsou to samé), tak nyní jsme schopni vyřešit i mnohoznačnost (díky spojení konceptů *Rusko* a *prezident* víme, že se v článku jedná o ruského prezidenta).

Cílem je tedy podle [18] detekovat významné témata a vytvořit efektivně a přesně odpovídající odkazy se správným významem. Wikipedia Miner Toolkit obsahuje algoritmy, které toto poskytují a které se skládají z těchto 3 základních kroků:

- Výběr kandidátů,
- Rozcestník odkazů,
- Detekce relevantních odkazů.

Vyběr kandidátů

V první fázi detekce témat se provede nashromáždění slov do n-pole. Následně se provede vyhodnocení, zda Anchor slovník (instance třídy Anchor viz výše) obsahuje některé z těchto



Obrázek 5.3: Článek rozšířený o dodatečné informace Wikipedie (čerpáno z [18])

slov. Tím se zjistí, které z nich Wikipedie zná. Je také důležité, aby nebyly brány v potaz slova jako *je* nebo *a*, které mnohdy nejsou použity jako relevantní termy. K odstranění takovýchto termů se vypočítává pravděpodobnost vytvoření odkazu na ně (čerpáno z [18]).

Rozcestník odkazů

Ve Wikipedii existuje samoučící se systém, který funguje za pomoci lidské odezvy, kde někdo musí manuálně vybrat správný cílový odkaz, který bude následně reprezentovat smysl v Anchor slovníku. Může ale nastat situace, že u některých slov bude více významů. Tady pak narážíme na problém dvojsmyslnosti.

To je vyřešeno v druhé fázi pomocí systému klasifikace, který je náchylný k přiřazení více používaných významů jednotlivých termů (pravděpodobněji přiřadí *prezidenta USA* než *prezidenta Marylandského senátu*). Kromě toho tento klasifikátor porovnává, jak je daný smysl příbuzný s okolními jednoznačnými termy, které v našem případě (obr 5.3) například zahrnují *Ruského prezidenta* a *Vladimíra Putina*. Díky tomu jsou pak vybrány odpovídající významy termů. (čerpáno z [18]

Detekce relevantních odkazů

Předešlé kroky vyprodukovaly množinu asociací mezi jednotlivými termy v dokumentu. Posledním úkolem je tedy potřeba určit, které z těchto témat (množin termů) jsou relevantní natolik, aby stály za vytvoření odpovídajících odkazů. Wikipedia poskytuje mnoho příkladů, jak toto provést.

Jedním z příkladů pro určení těchto relevantních odkazů může být užitečný například počet výskytů daného termu v n-poli z prvního kroku, protože čím víc je téma použito, tím více je pravděpodobné použití odkazu. Další způsob jakým lze provést určení, zda je potřeba odkazu na term je určení vzdálenosti mezi prvním a posledním výskytem termu v článku. Je to z toho důvodu, že důležité témata se objevují v úvodu, v závěru a také v celém dokumentu.

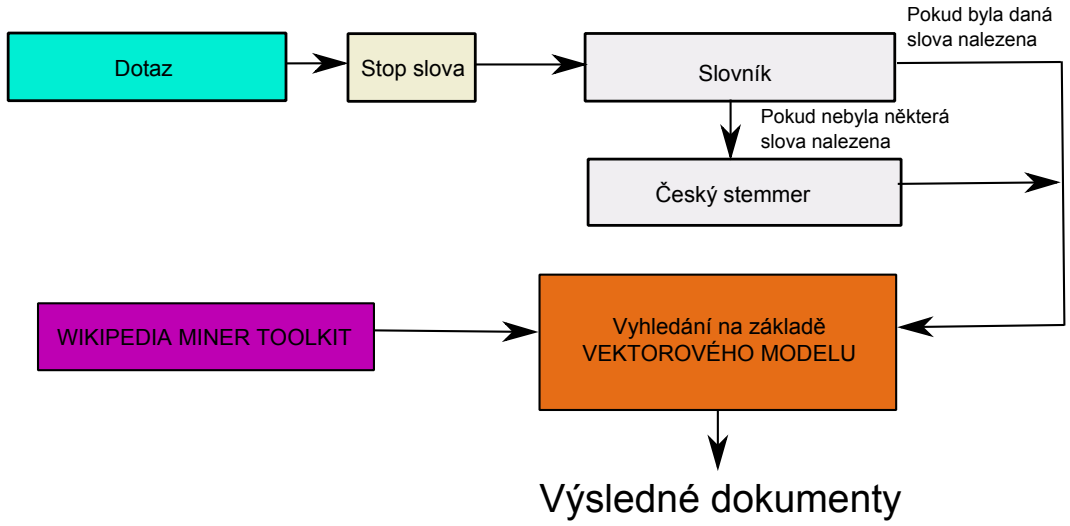
5.6 Shrnutí

V této kapitole byla rozebrána problématika získávání informací z wikipedie a výpočtu sémantické podobnosti. Podrobněji toto téma rozeberu v dalších kapitolách. Zájemce o detailní teoretický popis odkazuji na [18], kde jsou uvedeny další zdroje.

Kapitola 6

Návrh vyhledávacího systému

Samotnou aplikaci jsem se rozhodl navrhnout podle jednotlivých kroků v obrázku 6.1. V této kapitole je postupně rozeberu a popíši.



Obrázek 6.1: Návrh vyhledávacího systému pro dotazování v české Wikipedii

6.1 Stop slova

V první fázi se provede odstranění slov, nad kterými se nebude provádět indexování a následné vyhledávání. Jak již bylo řečeno výše, jde o tzv. stop slova. To jsou takové, které samy o sobě nemají žádný význam. Vetšinou jsou to spojky, předložky, některé zájmena a jiné. V našem případě budou muset být tyto slova doplněny o další a to o takové, které se vyskytují v podstatné části dokumentů wikipedie. Jejich indexováním bychom tak nezískali skoro žádné omezení výsledné množiny vyhledaných dokumentů. Musí se proto nezbytně provést analýza výskytu slov v jednotlivých dokumentech, která poukáže na příliš často se vyskytující slova. Ty pak budou také zařazena mezi již zmíněné stop slova.

6.2 Slovník

V druhé fázi se bude provádět vyhledávání slov ve slovníku. V této problematice se obecně používají různé konzolové programy, které kontrolují pravopis a umožňují také provést případné opravy. Mezi nejznámější zde patří **Ispell, Aspell, Myspell** či **Hunspell**. Ty také nyní porovnám a rozeberu jejich výhody a nevýhody.

Ispell

Pravděpodobně jde o nejstarší (vytvořen v roce 1971) a nejznámější program pro kontrolu pravopisných překlepů v elektronicky upravovaných textech. Písmeno I ve slově Ispell konkrétně vyjadřuje slovo International, tzn. že je daný nástroj použitelný pro maximální množství jazyků. Dnes používaná verze je založena na modifikaci programu z roku 1983, kde byl program přepsán v jazyce C. Samotný Ispell program používá 2 soubory a to samotný slovník slov a affix soubor ve kterém jsou popsány různé použití přípon ve slovníku.
Vlastnosti:

- podpora komprimovaných souborů,
- nižší nároky na paměť (v případě jedné instance programu),
- kvalitní podpora formátu TEX a LATEX,
- podpora formátu Nroff,
- o aktualizaci českého slovníku se stará správce (Petr Kolář) a tak je zajištěna jazyková čistota.
- podstatně horší inteligence návrhu oproti Aspellu
- velikost slovníku je do 10MB (neobsahuje tak velké množství slov jako Aspell)

Aspell

Druhým nejznámějším programem pro kontrolu pravopisů je Aspell. Ten vznikl s cílem nahradit Ispell.
Vlastnosti:

- vysoká inteligence návrhu oproti Ispell,
- dovede se učit z překlepů uživatele,
- má schopnost sdílet paměť mezi několika procesy,
- podpora UTF-8,
- oproti Ispellu umožňuje použít více slovníků.
- není znám manažer českého slovníku Aspell a tak může docházet k degeneraci slovníku a jeho nesprávným návrhům.

Myspell

Myspell je jednoduchý program pro kontrolu pravopisu, který používá affix kompresi. Jeho snahou je být co nejvíce kompatibilní s Ispellem.
Vlastnosti:

- dokáže pracovat s Ispell slovníky a aff soubory (po menší úpravě)
- nemá podporu pro přidávání slov do vlastních slovníků
- nemá podporu pro konverzi mezi různými formáty

Hunspell

Podle [19] jde o program pro kontrolu pravopisu, který je založen na Myspellu a je také kompatibilní s jeho slovníky. Tento program je primárně používán pro LibreOffice, OpenOffice.org, Mozilla Firefox 3, Thunderbird, či pro Google Chrome.
Vlastnosti:

- Rozšířená podpora pro jazykové zvláštnosti.
- podpora unicode (pravidla pracují s prvními 65535 unicode znaky)
- Hunspell provádí morfologickou analýzu a stemování
- podpora specifik jednotlivých jazyků (velké ostré s v němčině a jiné)
- umožňuje řešit podmíněné přípony, hononyma, zakázaná slova a jiné
- používá Myspell slovníky rozšířené o gramatické pravidla (dict soubor obsahuje slova a aff soubor obsahuje gramatické pravidla)

Shrnutí

Z výše uvedených rozborů vyplývá, že kontrolu pravopisu má nejlepší Hunpell. Tato skutečnost, ale také hodně závisí na kvalitě jednotlivých slovníků daného jazyka. Hunspell má velkou výhodu v podpoře specifik jednotlivých jazyků, v morfologických analýzách a jiné. Ispell v těchto ohledech oproti Hunspellu a Aspellu zaostává. Jeho jedinou výhodou v české variantě může být již zmíněná čistota slovníku.

Ačkoliv jsem zde porovnal jednotlivé slovníky, v PostgreSQL je aktuálně použitelná jen jedna varianta. Tento databázový systém totiž umožňuje použití jen Ispell slovníky. Podpora pro ostatní zde zatím není implementována. Moje finální volba proto byla usnadněna. V této části porovnání slovníků bylo čerpáno převážně z [37] a [19].

6.3 Morfologický analyzátor (stemmer)

Jak lze vidět na obrázku 6.1, pokud není dané slovo stop slovem a nebo nebylo nalezeno ve slovníku, provede se tzv. stemmování, které jak jsem již zmínil zde 2.3, se pokusí získat kořen daného slova. Nad výsledným slovem pak bude místo původního prováděno vyhledávání.

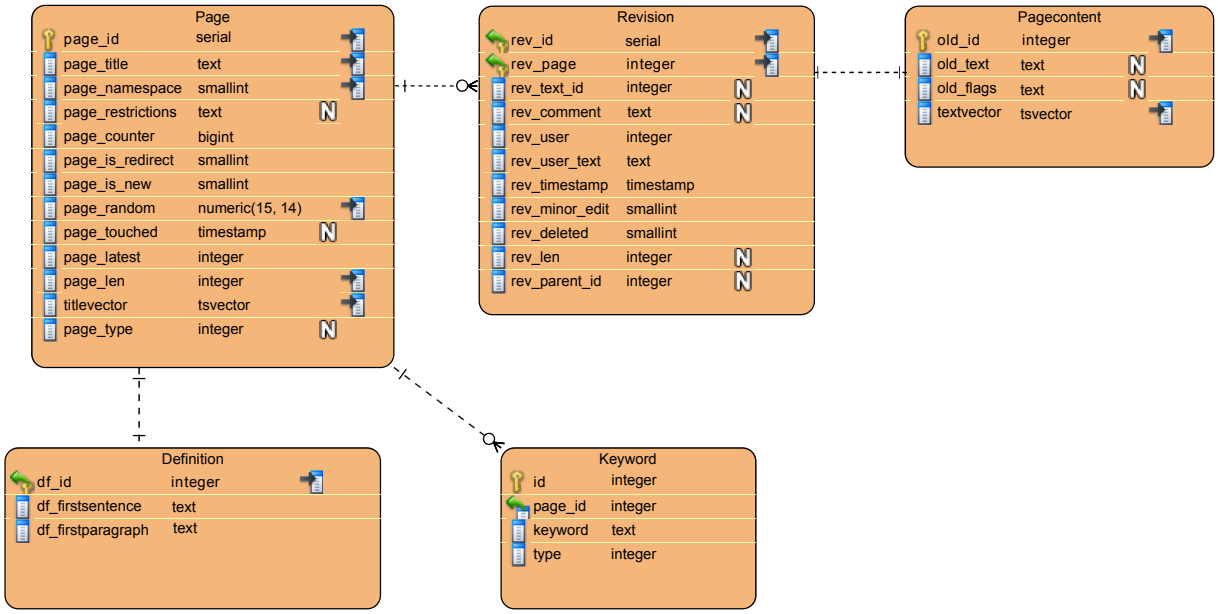
Pro tuto fázi jsem využil možnost použít Snowball stemmer, který vytvořil Ing. David Hellebrand ve své diplomové práci na VUT FIT. Podrobnější informace o této práci lze nalézt zde [11].

6.4 Vyhledávání

V této fázi návrhu jsme již zbaveni nežádoucích stop slov a nevhodných tvarů slov. Dle obrázku 6.1 tedy následuje samotné vyhledání slov.

Na obrázku 6.2 lze vidět tabulky ze kterých budou získávána data v jednotlivých cyklech vyhledávání. Hlavní jsou zde 3 tabulky Page, Revision a PageContent, jejichž struktura je dána mediawiki (viz 4.3), která je tzv. strojem Wikipedie. Proto musím z těchto tabulek vycházet. Z těchto tří tabulek je pak hlavní tabulka page, kde každý její záznam reprezentuje jeden článek. V této tabulce jsou zajímavé hlavně sloupce page_id, page_title a titlevector, nad kterým bude prováděno vyhledávání (viz kapitola implementace). V tabulce page_content je pak zajímavý hlavně sloupec textvector, nad kterým bude prováděno vyhledávání stejně jako u tabulky page. Nakonec budou vytvořeny dvě další - keyword a definition.

Tabulka keyword má sloužit k uložení klíčových slov (viz dále), aby nemusely být neustále vypočítávány pro každý článek znovu. Tabulka definition pak bude obsahovat první odstavec a první řádek každého článku tak, aby se při každém vyhledání nemusely procházet jednotlivé obsahy stránek. Z výše zmíněného popisu je tedy jasné, že obě tabulky mají za cíl urychlit vyhledávání, aby se nemusely provádět zbytečně opakované operace.



Obrázek 6.2: E-R model obsahující tabulky sloužící pro hlavní vyhledávání

6.5 Klíčová slova

Pro výpočet klíčových slov jsem se rozhodl použít nástroj Wikipedia Miner Toolkit, který jsem již zmínil v kapitole 5. Tam jsem také popsal tři kroky pro výpočet příbuznosti článků. Prvním krokem zde je **výběr kandidátů**, který se konkrétně provede pomocí metod **getLinksInIds a getLinksOutIdsAndCounts**. Tyto metody provedou identifikaci všech článků pomocí všech odchozích a příchozích linků odpovídajících článků.

Druhým a respektive třetím krokem byly **identifikace rozcestníku odkazů a detekce relevantních odkazů**. Tyto dva kroky jsou prováděny v metodě **getRelatednessTo**,

která provádí porovnání sémantické podobnosti dvou článků. Samotné výpočty jsou opět založeny jak na příchozích tak na odchozích odkazech každého článku. Následující vzorce jsou odvozeny ze zdrojových kódu Wikipedia Miner Toolkitu, které lze nalézt na přiloženém CD.

Pravděpodobnost z příchozích odkazů

Pro příchozí odkazy je pravděpodobnost příbuznosti článků získaná intuitivně tak, že se porovná počet stejných odkazů a různých odkazů obou článků. Z toho vyplývá, že:

$$\begin{aligned} a &= log(linksA) \\ b &= log(linksB) \\ ab &= log(linksBoth) \\ m &= log(ArticleCount) \end{aligned} \tag{6.1}$$

kde linksA vyjadřuje počet příchozích odkazů jednoho článku a linksB vyjadřuje počet příchozích odkazů druhého článku. Počet společných odkazů pak vyjadřuje hodnota linksBoth. Celkový počet článků je zde vyjádřen jako ArticleCount. Na základě výše zmíněných hodnot vypočteme výslednou příbuznost jako:

$$IN\_RelatednessA\_B = \frac{max(a,b) - ab}{m - min(a,b)} \tag{6.2}$$

Zde max(a,b) vybere větší z hodnot a a b. Naopak účel min(a,b) je přesně opačný.

Pravděpodobnost z odchozích odkazů

U odchozích linků je výpočet poněkud složitější. Využívají se zde principy vektorového modelu (viz kapitola 3.3), kde pro každý odkaz je vypočtena pravděpodobnost jeho použití. Mějme pole dataA a dataB, kde v každém z nich jsou id odkazovaných článků a počty odpovídajících odkazů pro daný článek (dále jako linkIDcount). Dále uvažujme vektory vectA a vectB do kterých se průběžně budou ukládat pravděpodobnosti výskytu jednotlivých odkazů daných článků. Pro jednotlivé dvojice článků idA a idB jako odkazů se pravděpodobnost přidaná do vectA vypočte jako:

$$probability\_idA = \begin{cases} log(\frac{ArticleCount}{linkIDAcount}), & idA = idB \\ log(\frac{ArticleCount}{linkIDAcount}), & (idA < idB \;\&\&\; idA > 0) \;||\; idB < 0 \\ 0, & jinak \end{cases} \tag{6.3}$$

Hodnota pravděpodobnosti použití článku idB jako odkazu se pak vypočte obdobně takto:

$$probability\_idB = \begin{cases} log(\frac{ArticleCount}{linkIDAcount}), & idA = idB \\ 0, & (idA < idB \;\&\&\; idA > 0) \;||\; idB < 0 \\ log(\frac{ArticleCount}{linkIDAcount}), & jinak \end{cases} \tag{6.4}$$

Po naplnění vektorů obou odkazovaných článků se provede výpočet úhlu mezi těmito vektory obdobně jako je to znázorněno na obrázku 3.3 v sekci o Vektorovém modelu. Mějme tedy dotProduct, magnitudeA a magnitudeB, které jsou jednotlivé směrnice. Vzorce pro výpočet těchto hodnot jsou pak následující:

$$maginitudeA \sim \sum_{i=1}^{t} vectA(i) * vectA(i), \tag{6.5}$$

$$maginitudeB \sim \sum_{i=1}^{t} vectB(i) * vectB(i), \tag{6.6}$$

$$dotProduct \sim \sum_{i=1}^{t} vectA(i) * vectB(i), \tag{6.7}$$

kde t je velikost z většího z vektorů vectA a vectB. Na základě těchto hodnot se na závěr vypočte výsledná příbuznost článků z odchozích linků jako:

$$OUT\_RelatednessA\_B = \frac{\frac{\pi}{2} - arc\ cos\frac{dotProduct}{magnitudeA\ *\ magnitudeB}}{\frac{\pi}{2}} \tag{6.8}$$

Celková sémantická příbuznost dvou článků se pak zprůměruje z obou získaných hodnot jako:

$$\frac{IN\_RelatednessA\_B\ +\ OUT\_RelatednessA\_B}{2} \tag{6.9}$$

.

6.6 Shrnutí

V této kapitole byl proveden rozbor a návrh jednotlivých plánovaných částí vyhledávacího systému. Bylo zde vyhodnoceno porovnání jednotlivých známých slovníků. Pro naše účely poslouží jen Ispell slovník, neboť je jako jediný podporovaný v PostgreSQL. Stěžejní částí této kapitoly je rozbor výpočtu sémantické příbuznosti jednotlivých klíčových slov. Celkový výpočet této příbuznosti bude podle vzorce 6.9 vypočten z odchozích a příchozích odkazů jednotlivých článků. Kromě této části zde byly popsány ostatní kroky jako filtrace stop slov, popis E-R modelu důležité části databáze a stemování, u kterého jsem se po domluvě s vedoucím rozhodl použít stemmer od Ing. Davida Hellebranda.

Kapitola 7

Implementace vyhledávacího systému

V této kapitole je popsána problematika zprovoznění jednotlivých komponent a programů potřebných ke spuštění a správnému fungování vyhledávácího systému. Jsou zde dostatečně provedeny popisy jednotlivých kroků s odkazy na detailní popis jednotlivých postupů. Dále je v této kapitole popsána implementace, úskalí na které jsem narazil a jejich řešení.

Zvoleným programovacím jazykem pro implementaci vyhledávání informací v české Wikipedii je jazyk **Java**. Vybraná databáze je **PostgreSQL**. Tato databáze je objektově-relačním databázovým systémem. Vybral jsem zde verzi 9.0, protože ta má již zabudované fulltextové vyhledávání pomocí **Tsearch2**. Jde o velice vyspělý a odzkoušený nástroj a proto jsem se jej rozhodl využít i v této práci. Původní operační systém, na kterém bylo vyvíjeno, byl Windows XP. Samotná verze mediawiki pomocí které běží wikipedie, pak byla zvolena 1.16. Pro zobrazování výsledků a interakci s uživatelem jsem použil technologie Java EE 6, GlassFish a Java Server Faces. Samotné vývojové prostředí jsem pak zvolil Netbeans. Celá práce probíhala na stroji Intel Core i5 CPU @2,68GHz s 2,5GB RAM, kde pro testování bylo provedeno navýšení RAM paměti na 10GB.

7.1 PostgreSQL a jeho komponenty

V průběhu práce jsem zjistil, že pro mé použití je nevhodná instalace PostgreSQL z binárních balíčků. Při této instalaci totiž nelze následně modifikovat části databázového systému nebo přidat vlastní kód. Tato možnost je pro tuto práci důležitá, neboť jsem potřeboval přidat vlastní stemmery. Z tohoto důvodu bylo nutné použít kompilovatelnou verzi PostgreSQL. Zprovoznit ji na systém Windows bylo značně problématické z důvodu složitosti vyhledání kompatibilních verzí knihoven a programů. Z tohoto důvodu a také z důvodu, že vetšina tutoriálů a návodů pro PostgreSQL je určena pro Linuxové systémy, jsem přešel na Linux, konkrétně na verzi Ubuntu 10.10. Zde je instalace ze zdrojových kódů v celku intuitivní. Postup instalace lze nalézt po stažení dané verze ze stránek [23] v kořenovém adresáři v dokumentu install. Před samotnou instalací je ale ještě nutné nainstalovat některé podpůrné knihovny a programy. V prostředí Ubuntu lze jejich instalaci provést jednoduše pomocí apt-get install. Pro podrobnější popis instalace a potřebných knihoven odkazuji zájemce na stránky [22].

Snowball

Jak jsem se již zmínil ve fázi návrhu, také jsem se rozhodl použít stemmer, který vytvořil Ing. David Hellebrand. Tento stemmer je vytvořen v jazyce Snowball (menší jazyk pro práci s řetězci [20]) a tudíž jej lze připojit k databázovému systému Postgres. Podrobněji se o tomto stemmeru mohou zájemci dočíst v Diplomové práci Ing. Hellebranda [11]. Zdrojové kódy stemmerů pro připojení lze nalézt na CD v sekci *source_files/stemmers*, kde soubory .c se vloží do zdrojových kódů PostgreSQL do adresáře */src/backend/snowball/libstemmer* a hlavičkové soubory .h do */src/include/snowball/*. Toto se provede buď před instalací celého PostgreSQL a nebo lze dodatečně zkompilovat jen snowball sekci pomocí Makefile. Vytvořený soubor *dict_snowball.so* je pak ale nutné zkopírovat do nainstalovaného databázového systému do adresáře */usr/local/pgsql/lib*.

7.2 Mediawiki a databáze

Jak již bylo řečeno výše Mediawiki je tzv. engine Wikipedie. V mém projektu její zprovoznění bylo hlavně z důvodu správného vygenerování databáze a možnosti zobrazení vrácených výsledků, když si chce uživatel zobrazit výsledný obsah vybrané stránky. Pro funkčnost mediawiki je nutné mít nainstalované PHP (nepoužívat verzi PHP 5.3.1), databázový server (v případě tohoto projektu PostgreSQL) a webovou službu (např Apache). Zdrojové kódy Médiawiki, podrobný návod na instalaci a zprovoznění lze pak nalézt zde [31].

Po nainstalování databáze jsem si ji naplnil daty. Ty lze získat ze stránek [32] ve formátu xml. Pro účely tohoto projektu postačí pages-articles.xml. Obsahuje data stránek, jejich obsah a propojení. Z tohoto souboru jsem si nejprve vytvořil sql soubor pomocí programu mwdumper, který lze nalezt na CD v adresáři *source_files/scripts*. Je nutné mít odpovídající verzi mwdumperu k dané Mediawiki, jinak může dojít ke špatnému zobrazování informací. V mém případě to byla verze mwdumper 1.16. Dalším krokem je import získaného sql souboru. Existuje více způsobů naimportování dat do PostgreSQL databáze (viz [14]), ale jediný funkční v mém případě je pomocí pgsql konzole. Důvodem je velký objem dat sql souboru (cca 1GB). Tento zmíněný postup je zapsán na CD v souboru *source_files/scripts/import_from_xml.bat*. Pro zajímavost ještě zmíním, že samotný import trval přes 6 hodin. Jednotlivé časy trvání importu lze vidět v tabulce 7.1

tabulka	doba importu	počet záznamů
page	1min 52 sec	311 554
pagecontent	6 hod 3 min 15 sec	311 554
revision	1min 22 sec	311 554

Obrázek 7.1: Časy importů do jednotlivých tabulek Wikipedie

7.3 Wikipedia Miner Toolkit

O tomto nástroji jsem se již zmínil v páté kapitole a v této práci hraje důležitou roli. Jeho zprovoznění je tedy nezbytné. Jedním z největších problémů při zprovoznění byla skutečnost, že dostupné verze tohoto toolkitu pracují s MySQL databází. Proto bylo prvním

krokem předělání zdrojových kódů, aby spolehlivě pracovaly s PostgreSQL databází. Dalším zádrhelem byla skutečnost, že samotný toolkit používá některé tabulky s obdobným obsahem jako Mediawiki. Konkrétně jde o tabulky page, redirect a categorylinks. Problémem zde byl fakt, že tyto tabulky nebylo vhodné spojit v jednu, jelikož samotný toolkit si do tabulky page ukládá také kategorie a rozcestníky, přičemž Mediawiki jen stránky. Z tohoto důvodu bylo nejlepším řešením zavést duplicitu a vytvořit tyto tabulky znovu. Vytvoření tabulek toolkitu a příkazů pro úpravu databáze se nachází na CD v souboru */source_files/sql/create_wiki_miner_db.sql*.

Po úpravě databáze je dalším krokem naplnění tabulek toolkitu. K vygenerování těchto dat opět postačí již zmíněný xml soubor. Z něj se nejprve pomocí skriptů toolkitu vygenerují potřebné csv soubory pro jednotlivé tabulky. Původní skripty obsahovaly chyby a sám autor David Milne přiznává, že perlovské skripty nejsou nejlepší variantou pro syntaxi Wikipedie, kde mnohdy dochází k zamrznutí nebo spadnutí skriptu. Autor sice již vytvořil novější verzi tohoto nástroje, který pro generování potřebných dat používá javu, ale tato verze ještě není mezi oficiálními verzemi ke stáhnutí a také se mi ji nepodařilo zprovoznit. Proto jsem se rozhodl upravit již zmíněné skripty pro novější verzi perlovských knihoven a také pro českou verzi wikipedie, tak aby fungovaly. Výsledné opravené skripty jsou uloženy na CD v adresáři *source_files/scripts/extraction*. Podrobný návod jak vyextrahovat data pro jednotlivé tabulky lze pak nalézt na CD v souboru *source_files/wikipedia − miner_1.1/readme.htm*. Samotná extrakce dat je časově náročný proces. Doba provedení této operace byla zhruba 10 hodin.

7.4 Stop Slova

Výběr správných stop slov efektivně urychlí vyhledávání a jak jsem se již zmínil ve fázi návrhu, kromě známých stop slov jako je například většina spojek nebo zájmen se musí získat slova používaná v podstatném množství článků.

Pro zjištění těchto slov jsem poprvé využil výhody Tsearch2. Jak jsem se již zmínil výše existuje zde funkce **ts_stat**, která umožňuje provádět statistiky nad jednotlivými sloupci tabulky databáze. Tyto sloupce musí být typu **tsvector** (jde o tříděný seznam různých lexemů, což jsou normalizovaná slova). Tento vektor je pak většinou používán s textovým sloupcem nad kterým chceme provádět rychlé textové operace (hlavně vyhledávání viz [21]). Použití vyhledávání nad tímto sloupcem místo nad textovým sloupcem tyto operace podstatně urychlí. Každý řádek tabulky, kterou vrací funkce ts_stat, obsahuje tři sloupce a to **word, nentry a ndoc**. Sloupec **word** obsahuje samotný lexém, sloupec **nentry** pak obsahuje celkový počet výskytu daného slova. Stěžejní význam pro mou práci má sloupec **ndoc**. Ten totiž vrací pro každé slovo počet dokumentů ve kterých se vyskytuje. V mém případě šlo konkrétně o dotaz nad tabulkou *pagecontent*, kde jsou uloženy obsahy jednotlivých stránek.

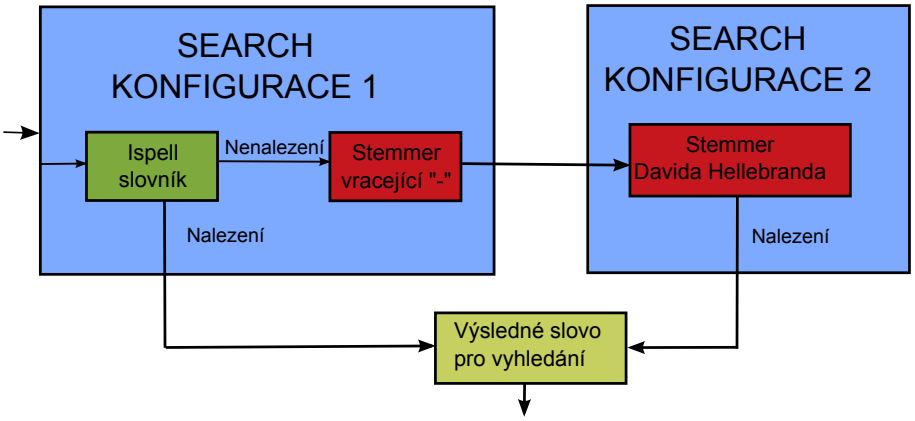
Po seřazení dle počtu výskytu jednotlivých slov ve stránkách jsem pak byl schopen vybrat nejpoužívanější slova a zařadit je mezi stop slova. Původně bylo mým záměrem toto vytvoření stop slov zahrnout do inicializace v programu. Z důvodu výskytu důležitých slov v daném seznamu jsem ale musel od tohoto nápadu upustit a upravit daný seznam ručně. Pro zajímavost pak mohu zmínit, že v mém případě trvalo provedení samotného dotazu provedeného funkcí *ts_stat* nad 311 596 položkami tabulky 3 minuty a 20 sekund. Podrobnější informace o fulltextovém vyhledávání v PostgreSQL databázi lze nalézt zde [21].

7.5 Slovník a stemmer

Prvním krokem ve zprovoznění požadovaných slovníků je připojení těchto slovníků a stemmerů do databázového systému. V PostgreSQL již sice nějaké slovníky a snowballové stemmery existují, ale ani jeden z nich není pro češtinu. Pro jejich připojení k databázi jsem tedy použil příkaz **CREATE TEXT SEARCH DICTIONARY**. K jeho provedení je potřeba mít umístěny soubory Ispell slovníku (konkrétně soubory s příponou dict, aff a stop, jenž obsahuje předem získané stop slova) v příslušných adresářích. Následně vytvořené slovníky lze používat pro vyhledávání a stemmování ve funkcích PostgreSQL.

Dalším krokem bylo vytvoření tzv. vyhledávacích konfigurací, které umožňují řazení slovníků a stemmerů za sebou. Tato konfigurace funguje tak, že postupným procházením jednotlivých slovníků, lze hledat dané slova. V případě nalezení slova se již neprovádí hledání v dalších slovnících konfigurace. Tím lze jednoduše určit prioritu jednotlivých slovníků a stemmerů.

V mém případě bylo nutností použít dvě vyhledávací konfigurace. První z nich obsahuje Ispell slovník a jednoduchý stemmer, který bude vždy vracet pomlčku. Druhá konfigurace pak obsahuje stemmer od Ing. Davida Hellebranda viz [11]. Toto rozdělení do dvou běhů a vytvoření jednoduchého stemmeru pro první konfiguraci bylo z důvodu rozeznání stop slov od slov určených ke stemmování. Slovník totiž stop slova ignoruje, ale stemmer by s nimi dále pracoval, což by byla chyba. Pro lepší pochopení lze tuto část vyhledávání pomocí konfigurací vidět na obrázku. 7.2.



Obrázek 7.2: Vyhledávání ve slovnících a stemmerech pomocí vytvořených konfigurací

7.6 Vyhledávání informací

Nyní máme k dispozici slova, která jsou již zbavena stop slov a případně nad nimi bylo provedeno stemmování. Dalším krokem tedy bude vyhledávání relevantních dokumentů v databázi Wikipedie.

Samotné vyhledávání v PostgreSQL musí být prováděno stejně jako ve většině běžných vyhledávacích metodách nad nějakou reprezentací dokumentu (viz sekce 2.3). Ta musí být obecně taková, aby nad ní bylo možno provádět rychlé zpracování a vyhledávání. V PostgreSQL k tomuto účelu slouží již zmíněný datový typ tsvector, jenž je speciálním vektorem slov a jejich pozic. Tyto hodnoty označují, na které pozici se konkrétní slovo v textu vyskytuje. Například věta "Dneska je krásně"bude v tsvectoru uložena jako 'dneska':1 'je':2

'krásně':3. K vytvoření tohoto vektoru z textu pak slouží funkce **to_tsvector**.

Pokud již máme takto vytvořenou reprezentaci dokumentů, můžeme provádět konkrétní vyhledávání. K tomu zase poslouží funkce **tsquery**, která z požadovaného dotazu (řetězce slov) vytvoří reprezentaci, jenž je potřebná k dotazování nad tsvectorem. Tento řetězec je vytvořen kombinací jednotlivých slov a operátorů.

Nyní již máme reprezentace potřebné k vyhledávání textu. Samotné fulltextové vyhledávání v PostgreSQL se provádí pomocí operátoru **@@**. Zmíněný operátor @@ tedy očekává nalevo dokument, který je reprezentován tsvectorem a napravo dotazovaný text reprezentovaný pomocí tsquery. V mém případě může dotaz nad názvy stránek vypadat následovně:
Dotaz1:

```
SELECT page_id FROM mediawiki.page
WHERE (titlevector @@ (to_tsquery('les | polesí & !voda')))
```

Zde titlevector je tsvector, který je sloupcem v tabulce stránek a který obsahuje název konkrétních stránek. Text v tsquery pak vyjadřuje, že požadujeme, aby se v daném vektoru vyskytovalo slovo *les* nebo *polesí*, ale aby se zde také nevyskytovalo slovo *voda*. Pokud pak konkrétní tsvector tabulky splňuje daný výraz definovaný pomocí tsquery, pak dotaz vrací true a celý řádek tabulky je případně navrácen dotazem. Pro podrobnější a rozsáhlejší popis odkazuji zájemce do dokumentace [21].

Tímto zmíněným postupem jsem vyhledával názvy stránek, které obsahují alespoň jedno klíčové slovo a zároveň neobsahují žádné s vykřičníkem před daným slovem. Dále jsem se rozhodl vyhledat dokumenty nejen v názvu jednotlivých stránek, ale také v obsahu stránek. Zde dotaz opět vrací stránky, které obsahují alespoň jeden z požadovaných termů.
Dotaz2:

```
SELECT r.rev_page FROM mediawiki.revision r RIGHT JOIN
  (SELECT old_id FROM mediawiki.pagecontent
   WHERE (textvector @@ (to_tsquery('les | polesí & !voda')))) pc
   ON r.rev_id=pc.old_id)
```

Nyní máme hodnoty id článků vyhledané nad názvy stránek a pak také nad obsahem stránek. Spojení těchto dvou množin se provede pomocí klausule **UNION** (viz Dotaz3), která vrátí id článku, pokud se nachází alespoň v jedné z množin. Původně bylo spojení množin provedeno pomocí OR, ale pro tento konkrétní případ bylo méně efektivní. Důvodem je skutečnost, že UNION oproti OR používá indexy, což při tak velkých tabulkách nad jakými dotazy probíhají podstatně urychlí operaci.

7.6.1 Hodnocení relevantnosti

V této fázi jsou vyhledány všechny dokumenty, které obsahují požadované termy v názvu nebo obsahu stránky. Těchto výsledných dokumentů může být enormní množství, které mnohdy není relevantní. Z tohoto důvodu by bylo vhodnější vrátit uživateli získané stránky v pořadí podle relevantnosti. V PostgreSQL existují dvě funkce pro výpočet relevantnosti dokumentu vůči dotazu a to **ts_rank a ts_rank_cd**.

ts_rank

V sekci 2 jsem již podrobně rozebral jednotlivé nejznámější modely vyhledávání a ve shrnutí jsem následně vyhodnotil jako nejvhodnější vektorový model, který vypočítá na základě směrnic dokumentu a dotazu relevantnost dotazu (viz 3.3). Na tomto principu je také

založena funkce ts_rank a proto ji nebudu podrobněji teoreticky rozebírat. Mohu zmínit, že hlavní důraz je kladen na tf faktor (viz 3.3). Nad samotným výsledkem této funkce je standartně prováděna normalizace založená na délce dokumentu.

ts_rank_cd

Tato funkce je oproti ts_rank založena na metodě **Cover Density Ranking** (dále jako CDR). Hlavní myšlenka této metody je podobná jako u vektorového modelu. Rozdíl je zde v tom, že CDR klade důraz na blízkost jednotlivých termů v dokumentu. Je proto nutná znalost pozic jednotlivých slov v dokumentu. Metoda CDR vypočítává relevantnost dokumentu ve dvou základních krocích:

1. Nejdříve jsou jednotlivé stránky (dokumenty) obsahující jeden nebo více termů dotazu ohodnoceny, podle toho kolik obsahují různých termů daného dotazu. Na základě tohoto ohodnocení jsou dokumenty rozděleny do tzv. koordinačních úrovní (coordination level). V praxi to může vypadat tak, že dokumenty obsahující jen jeden požadovaný term budou mít ohodnocení normalizováno do rozsahu (0,1], dokumenty obsahující dva různé termy budou ohodnoceny v rozsahu (1,2] a tak dále.

2. V druhém kroku se pak pro dokumenty v jednotlivých koordinačních úrovních vypočítá celková relevance následovně. Mějme množinu cover obsahující dvojice různých dvou termů (p_j, q_j), kde p_j je pozice prvního termu a q_j je pozice druhého termu. Tato dvojice pak určuje nejmenší vzdálenost těchto dvou termů. Předpokládá se, že g_j je větší než p_j. Výsledná hodnota pro cover množinu daného dokumentu je spočtena následovně:

$$S(w) = \sum_{j=1}^{n} I(p_j, q_j) \tag{8.1}$$

kde

$$I(p_j, q_j) = \begin{cases} \frac{\lambda}{g_j - p_j + 1}, & pokud q_j - p_j + 1 > \lambda \\ 1, & jinak \end{cases} \tag{8.2}$$

λ je v tomto vzorci konstantou o hodnotě 16, která byla určena podle [36] na základě testů jako optimální. Jak lze vidět, tímto výpočtem tak budou penalizovány dokumenty s termy, které jsou daleko od sebe.

Obě tyto funkce mají stejné parametry:

```
([ weights float4[], ] vector tsvector, query tsquery
[, normalization integer ]) returns float4
```

kde:

- weights - je pole, které umožňuje provést nastavení priority jednotlivých částí (většinou sloupců tabulky)
- tsvector - reprezentuje dokument (viz výše)
- tsquery - reprezentuje uživatelský dotaz (viz výše)
- normalization - jde o pole, které umožňuje normalizovat výsledek na základě délky dokumentů, vzdálenosti termů a jiné (pro podrobnější popis odkazuji na [21])

Povinné jsou zde jen druhý a třetí parametr. Pomocí těchto funkcí jsem provedl sadu několika testů, kde výsledky byly u obou funkcí většinou obdobné. Nakonec jsem z těchto dvou funkcí vybral ts_rank a to ze dvou důvodů. Prvním důvodem je skutečnost, že ts_rank_cd rozděluje výsledky do skupin podle počtu nalezených slov dotazu a vypočítává jejich vzdálenost. To je nežádoucí například v případě hledání dotazu 'česká řeka'. Po zpracování slovníkem jsou totiž hledána slova 'český česká řeka'. Budou tak výše hodnoceny dokumenty obsahující slova česká a český nikoliv dokumenty obsahující slovo řeka. Druhým důvodem jsou testy v této publikaci [3], které hodnotí funkci ts_rank jako lepší.

Vybraná funkce mi tak umožnila provést vyhodnocení relevantnosti jednotlivých řádků tabulky vůči termům v tsquery. Na základě tohoto vyhodnocení se provede seřazení stránek pomocí klausule **ORDER BY**.
Výsledný dotaz vzniklý spojením z předešlých vypadá takto:
Dotaz3:

```
SELECT pa_re.page_id, pa_re.page_title,
ts_rank(pc.textvector,(to_tsquery('fotbal')))+
ts_rank(pa_re.titlevector,(to_tsquery('fotbal')))*10 as totalrank
FROM (mediawiki.page pa LEFT JOIN mediawiki.revision re
  ON pa.page_id=re.rev_page) pa_re
  LEFT JOIN mediawiki.pagecontent pc
    ON pa_re.rev_id=pc.old_id
WHERE pa_re.page_id in
  Dotaz1
UNION
  Dotaz2
ORDER BY totalrank;
```

Jak lze vidět hodnota výpočtu relevantnost nad názvem stránky je vynásobena hodnotou 10. To jsem provedl z toho důvodu, že považuji více relevantní ty dokumenty, ve kterých se dané slova vyskytují v jejich názvech. Ve výsledném programu lze ale tuto hodnotu nastavovat uživatelem. Samotný program pak umožňuje provádět výhledávání nejen v obou tabulkách, ale také jen v obsahu a nebo v názvech stránek (viz dále). Kompletní dotazy lze pak nalézt na CD v souboru *source_files/sql/search.sql*. V této kapitole bylo čerpáno z [21] a [36]

7.6.2 Generalized Inverted Index (Gin index)

I když je používáno speciální fultextové vyhledávání pomocí operátoru @@ nad tsvectorem, i přesto je vyhledávání zejména nad obsahem stránek velice pomalé a neefektivní. Řádově zde vyhledávání trvá až desítky sekund (viz tabulka 7.3). Bylo tedy nutné provést další urychlení. Přesně k tomuto účelu u fulltextového vyhledávání lze použít tzv. Gin index. Ty fungují nad sloupcem typu tsvector, kde vytvoří množinu párů. V tomto páru je první slovo a druhé je seznam ID řádků, ve kterých se dané slovo nachází. Tak lze provést podstatně rychlejší vyhledání, kde jsou rovnou vráceny seznamy ID pro jednotlivé slova. Pro zajímavost mohu zmínit, že vytvoření samotného GIN indexu nad tabulkou názvů stránek trvalo necelých 6 sekund a nad tabulkou obsahu stránek pak 1 hodinu a 10 minut. Tento extrémní rozdíl byl bezesporu způsoben obrovským množství slov v obsahu stránek, které musel PostgreSQL naindexovat. V této fázi jsem narazil na problém, že GIN indexy potřebují ke své činnost velké množství paměti. Pomocí klauzule EXPLAIN ANALYZE jsem zde nejdříve

zjistil, že nejvíce času skutečně zabralo opět vyhledávání (konkrétně operace hash join) a funkčnost GIN indexu byla neefektivní. Provedl jsem tedy navýšení RAM paměti z 2,5GB na 10GB a dotazy skutečně zaznamenaly podstatné zrychlení. Výsledky trvání dotazů ve všech fázích lze pak opět vidět v tabulce 7.3.

Obrázek 7.3: Tabulka hodnocení rychlosti jednotlivých dotazů v různých fázích

dotaz v tsquery	počet řádků	bez GIN indexu	s GIN indexem	po použití UNION a JOIN
fotbal	3278	4214 ms	1544 ms	596 ms
test	1300	4119 ms	900 ms	308 ms
školní povinnosti	2171	4190 ms	1215 ms	435 ms
obyvatelé města	34899	22305 ms	10302 ms	7338 ms
foto	19152	9803 ms	4633 ms	1015 ms
letech	29643	5246 ms	4821 ms	5432 ms

7.6.3 Seq scan vs. Bitmap heap scan

V tabulce 7.3 lze vidět, že i přes použití GIN indexů a po vylepšení pomocí klauzulí UNION a JOIN u dotazů se slovy 'letech' nebo 'obyvatelé města' trvalo vyhledávání podstatně déle než u ostatních testovaných výrazů. Po pár testech jsem zjistil, že je to způsobeno tím, že u těchto výrazu bylo použito pro prohledání tabulky **sekvenčního scanu** (dále jako SS). U ostatních testovaných výrazů byla pro prohledání naopak použita metoda **Bitmap heap scan** (dále jako BHS). Je proto logické, že SS oproti BHS trvá podstatně déle, neboť prochází postupně všechny záznamy. Otázka tedy zní: Proč byl použit SS místo BHS. Po pár testech jsem došel ke zjištění, že když Postgres planner (plánovač, který rozhodne, jak se bude daný dotaz provádět) zjistí vrácení velkého poměru záznamů, pak použije SS. Obecně je použití SS efektivnější v těch případech, kdy máme tabulku s menším množstvím dat. V mém případě je ale efektivnější použít BHS ze dvou důvodů. Prvním z nich je fakt, že v tabulce se nachází velké množství dat (pro které je lepší BHS). Druhým důvodem je skutečnost, že je malá pravděpodobnost, aby docházelo k dotazům, které by vracely podstatnou většinu dokumentů, pro které je SS výhodnější. Jednalo by se totiž buď o stop slova, nad kterými neprobíhá vyhledávání nebo by muselo být zadáno velké množství slov, kde každé z nich se vyskytuje ve významné části dokumentů. Dále také očekávám, že pro vyhledávání v encyklopedii jako je Wikipedie nebude uživatel zadávat dlouhé sekvence slov.

Na základě těchto skutečností jsem se rozhodl, že provedu vypnutí Sekvenčního scanu, čímž automaticky upřednostním Bitmap heap scan. To se provede pomocí příkazu:

$$SET\ enable\_seqscan = OFF$$

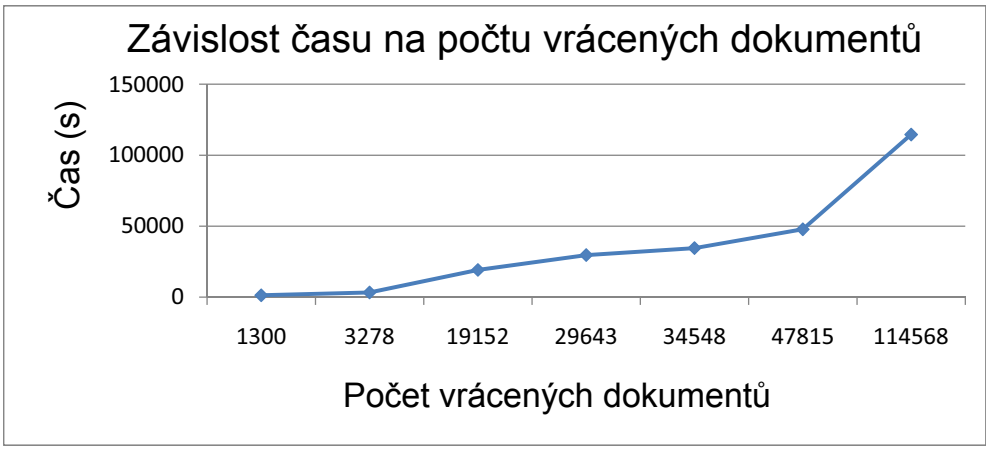
Po vypnutí sekvenčního scanu se doba trvání u všech dotazů, které vrácely velké množství záznamů, podstatně zlepšila. Otestoval jsem nejen dotazy obsahující slova, které se nachází na hranici seznamu stop slov (slovo letech), ale také slova, která měla správně do seznamu stop slov patřit (slova český a narození). Z důvodu důležitosti těchto slov jsem je ze zmíněného seznamu odstranil. Dále jsem také zkusil otestovat sekvence více slov, které se vyskytují ve větším množství dokumentů. Samotné časy pak dosahovaly maximálně v jednotkách sekund, což je přijatelné. Tímto testem jsem dokázal, že vypnutí sekvenčního skenu nebude mít žádné vedlejší nežádoucí účinky. Konkrétní časy zmíněných dotazů lze vidět v tabulce 7.4.

Obrázek 7.4: Porovnání dotazů při použití seq scan vs. bitmap heap scan

dotaz v tsquery	počet řádků	seq scan	bitmap heap scan
letech	29643	5432 ms	1384 ms
český	23421	5200 ms	1037 ms
narození	34548	5532 ms	2056 ms
obyvatelé města	34899	6025 ms	2123 ms

7.6.4 Limit

Poslední úpravu dotazu, kterou provedu, je omezení počtu vrácených dokumentů z tabulky pagecontent. Může totiž dojít k situaci, kdy uživatel zadá větší množství slov, které se nachází na hranici seznamu stop slov. Dotaz by pak vracel velké množství (řádově desítky tisíc) mnohdy nerelevantních dokumentů. Kromě toho by se výrazně navýšila doba trvání samotného dotazu (viz graf 7.5). Toto navýšení způsobují jednotlivé LEFT JOIN klausule, které s vyšším počtem výsledků vrácených z tabulky pagecontent trvají déle.



Obrázek 7.5: Závislost času na počtu vrácených dokumentů

Z těchto důvodů jsem použil klausuli LIMIT, která mi umožní omezení počtu vrácených dokumentů. Tuto hodnotu maximálního počtu vrácených výsledků si může uživatel nastavit v parametrech programu.

7.7 Klíčová slova

Poté, co jsem získal výše zmíněné články (dokumenty), přejdu k dalšímu kroku. Tím je provedení získání jednotlivých klíčových slov ke každému z těchto článků. Uživatel pak bude moci díky jejich přítomnosti lépe pochopit obsahový význam daného článku a také si bude moci vybrat další směr vyhledávání.

K tomuto kroku nyní konečně poslouží výše zmíněný nástroj Wikipedia Miner Toolkit, který jsem popsal v kapitole 5 a zmínil jsem také jeho možnost použití v návrhu. Základem pro výpočet klíčových slov je zde mít naplněné tabulky pro odchozí a příchozí odkazy jednotlivých článku na jiné (tento krok již musel být proveden při zprovoznění toolkitu

podle postupu v kapitole 7.3). Je to z toho důvodu, že jednotlivé klíčové slova se získávají právě z těchto odkazu.

V metodě **setAllKeyWords** je implementováno samotné získávání klíčových slov a jejich uložení do tabulky keyword. Postupně se zde pro každý článek vyhodnotí sémantická příbuznost všech jeho odkazovaných článků (dále jako klíčová slova). Tento výpočet se provede pomocí metody **getRelatednessTo**, jejíž výpočty jsem podrobně popsal v návrhu. Jednotlivé klíčové slova jsem následně seřadil dle hodnoty relevance, kterou daná metoda vrací. Po pár testech jsem zjistil, že únosné množství klíčových slov se převážně pohybuje okolo 20% - 30% příbuznosti. Na druhou stranu jsem ale zjistil, že u velké části článků, je příbuznost většiny klíčových slov do 10%. Rozhodl jsem se tedy, že rozdělím klíčová slova do třech skupin a nechám výběr úrovně jejich sémantické příbuznosti vůči článku na uživateli. Vznikly tak 3 skupiny klíčových slov, kde první je s příbuzností nad 30%, druhá je s příbuzností mezi 20%-30% a třetí jsou zbylá klíčová slova. Rozdělení jednotlivých klíčových slov do těchto skupin je zajištěno druhým a třetím parametrem metody setAllKeywords. Pro zajímavost mohu uvést, že celková doba extrakce klíčových slov byla 54 minut a 27 sekund. Po extrakci klíčových slov a jejich naplnění do tabulky je lze jednoduše získávat na základě identifikátoru daného článku.

7.8 Zprovoznění programu

Po zprovoznění všech potřebných programů a knihoven, lze přejít ke zprovoznění aplikace pro vyhledávání systému.

7.8.1 Technologie pro zobrazení

Samotný program využívá pro zobrazení a interakci s uživatelem platformu Java EE 6 spolu s Java Server Faces. Tyto technologie jsem zvolil z těchto důvodů:

1. Umožňují oddělení datového modelu, uživatelského rozhraní a řídící logiky do tří nezávislých komponent (tzv. Model-View-Controller). Pokud tedy bude program na serveru, uživateli se budou zobrazovat jen výsledná data pro zobrazení a nebudou se tedy na straně klienta provádět žádné výpočty
2. Vývoj pomocí těchto technologii je relativně jednoduchý a rychlý.
3. Tyto technologie jsou vysoce distribuované, přenosné a rychlé.

7.8.2 Parametry v programu

Vytvořený program se nachází na CD v adresáři *source_files/wiki*. Nejdříve je nutné při prvním spuštění provést nastavení následujících položek. V souboru *StopLists.java* je nezbytné nastavit zdrojové a cílové adresáře pro seznamy slov a také názvy souborů obsahujících tyto seznamy. Ve volání funkce *ReadFileList* se musí také nastavit v třetím parametru počet vstupních stop listů. V programu pak budou tyto listy porovnány a slova, která se vyskytují minimálně ve dvou seznamech budou použita do výsledného seznamu slov. Tím bude zajištěno, že pokud existuje více dostupných seznamů slov, pak se přidají jen ta slova, která jsou skutečně za stop slova považována. Tuto funkcionalitu jsem zavedl z toho důvodu, že jsem nalezl větší množství dostupných seznamů, které se částečně liší. Skutečnost, že dané slovo se nachází ve dvou souborech, pak považuji za dostatečný důvod pro přidání do výsledného seznamu. Po vytvoření tohoto souboru slov jsou následně přidány stop

slova získané z Wikipedie (viz sekce 8.2). Další nutnou položkou nastavení je v souboru *PagesController.java*, kde se musí nastavit adresa serveru na kterém běží mediawiki. Tato hodnota se zadá do parametru **serverAdress**.

7.9 Zobrazení dat a interakce s uživatelem

Výsledky v podobě článků a klíčových slov se zobrazí uživateli (ukázka je na obrázku 7.6), který může na základě nich zobrazit výsledné články pomocí mediawiki a nebo může pomocí nich vyhledávat dále. Další vyhledávání na základě klíčových slov lze provést pomocí tlačítek plus a mínus u každého z článků a jeho klíčových slov. Po kliknutí na některé z těchto tlačítek se odpovídající název přidá do sezamu. Rozdíl mezi tlačítky plus a mínus je v tom, že po kliknutí na plus, bude název požadován ve výsledku. Naopak po kliknutí na tlačítko mínus se před slovo vloží vykřičník, což značí, že nechceme aby dané slovo obsahovaly výsledné články. Další vyhledání pak bude probíhat nejen na základě zadaných slov, ale také na základě vybraných klíčových slov (ukázku přidaných slov lze vidět na obrázku 7.8). Takto lze vyhledávat dokud uživatel nenalezne požadovanou stránku. Po kliknutí na konkrétní název se zobrazí nový panel s výslednou stránkou.

Obrázek 7.6: Zobrazení výsledků uživateli

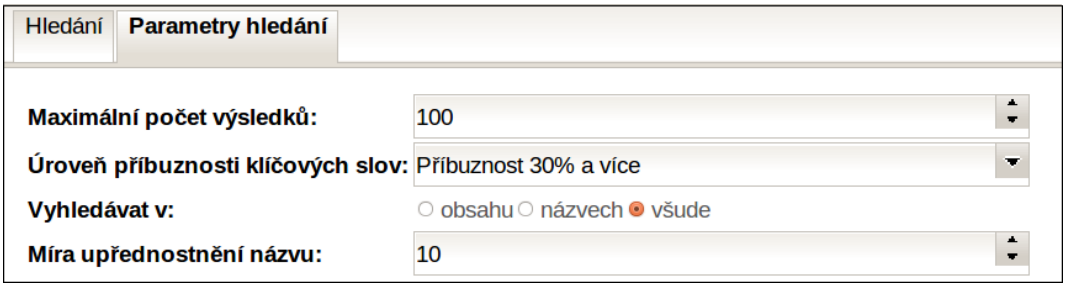
Daný program kromě výše zmíněné funkčnosti nabízí různé nastavení:

1. Lze nastavit maximální počet výsledků, aby se uživatel v případě vrácení extrémního množství článků nemusel probírat velkým množstvím výsledků. Výchozí hodnota je zde 100 položek. Jednotlivé výsledky jsou ale i přesto stránkovány, kde na každé stránce je pro lepší přehlednost maximálně 20 výsledků.

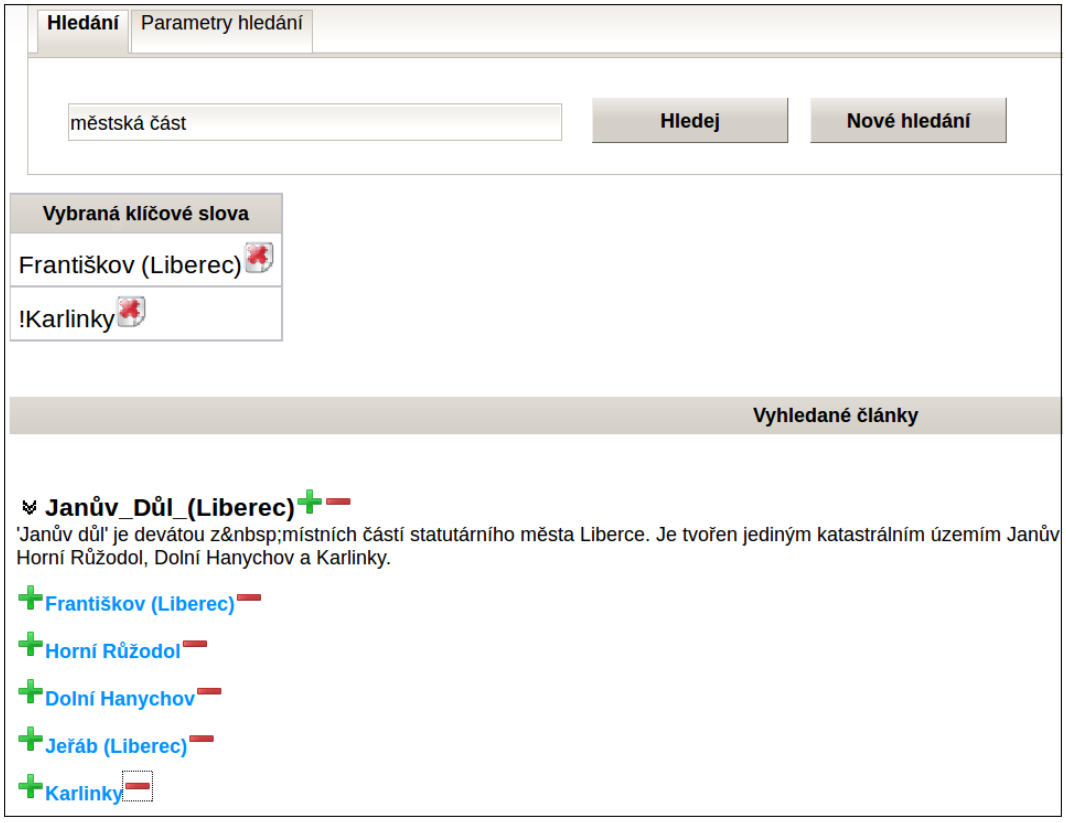
2. Dále lze nastavit úroveň sémantické příbuznosti klíčových slov vůči článku. Může totiž docházet k situacím, kdy článek má klíčové slova jen do 20% nebo dokonce do 10% sémantické příbuznosti. Z toho důvodu lze zadat tuto úroveň v závislosti na konkrétní situaci. Výchozí hodnota je zde 30% a více.

3. Dalším z užitečných parametrů nastavení je výběr, zda chceme vyhledávat jen v názvech článku, v obsahu či v obojí. Výchozí hodnota je zde nastavena na vyhledávání v obsahu a také v názvech.

4. V neposlední řadě lze nastavit míru upřednostnění názvu. Jde o hodnotu, kterou se vynásobí ohodnocení relevantnosti názvu stránky. Tím si může uživatel určit míru upřednostnění názvu stránky oproti obsahu. Tato položka má efekt jen tehdy, pokud se vyhledává v obsahu a také v názvech.

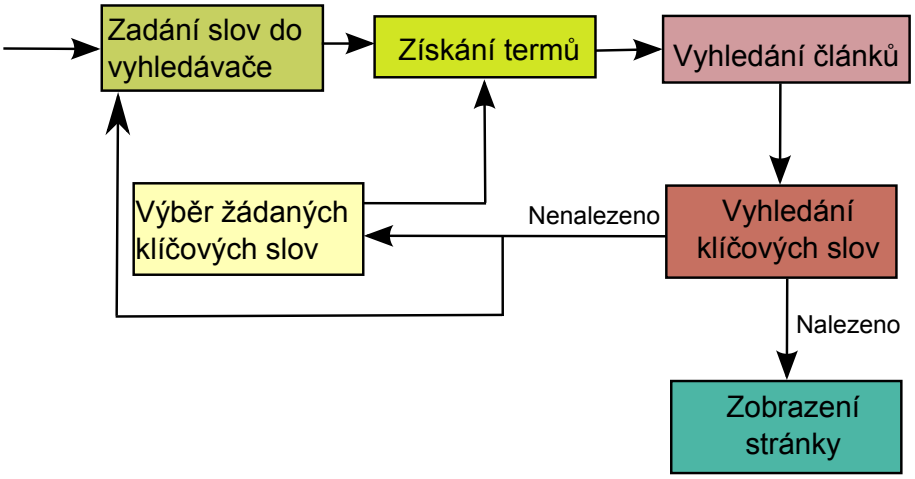


Obrázek 7.7: Nastavení programu



Obrázek 7.8: Výběr klíčových slov pro další vyhledání

Samotný program pak může probíhat podle výše uvedeného postupu v cyklech dokud uživatel nenalezne požadovaný dokument. Zjednodušený diagram tohoto cyklu lze pak vidět na obrázku 7.9.



Obrázek 7.9: Cyklus programu

Kapitola 8

Testování

Tato kapitola se zabývá hodnocením implementovaného algoritmu vyhledávání.

8.1 Hodnocení algoritmu vyhledávání

Pro hodnocení kvality vyhledávání se nejčastěji používají následující metriky.

Recall a Precision

Recall udává poměr množství nalezených relevatních dokumentů ke všem relevantním dokumentům. Vzorcem lze pak zapsat:

$$Recall = \frac{Relevantni\ vyhledane}{Relevantni\ dokumenty} \tag{9.1}$$

Precision oproti tomu udává poměr relevantních nalezených dokumentů ke všem vyhledaným dokumentům

$$Precision = \frac{Relevantni\ vyhledane}{Vsechny\ vyhledane} \tag{9.2}$$

Tyto hodnoty většinou není jednoduché získat a ve většině případů se odhadují. Zvláště metrika Recall je problématická pro výpočet, neboť mnohdy nejsme schopni určit skutečný počet relevantních dokumentů v celé databázi.

Mean average precision

Jde o takovou metodu, která umožní vybrat vzorky vyhledaných dokumentů a z těch provést vyhodnocení relevantnosti. Tato metrika hodnotí, jak velká část vyhledaných dokumentů je opravdu relevantní a v neposlední řadě bere také v úvahu pořadí jednotlivých vrácených dokumentů. Může zde pak dojít k situaci, že pro stejnou množinu získáme jinou hodnotu MAP (viz [35]). Pro definici Mean average precision (dále jako MAP) nejdříve nadefinuji metriku **Average precision**. Tato hodnota určuje přesnost metody pro každý relevantní vrácený dokument v pořadí. Pro lepší pochopení uvedu příklad. Mějme 4 vrácené dokumenty, kde relevantní jsou první, druhý a čtvrtý. Dále mějme dvě hodnoty **Hits** (aktuální počet dokumentů) a **Relevant hits** (počet relevantních dokumentů v dosud zpracovaných). Výpočet pak probíhá tak, že bereme postupně výpočet pro každý relevantní dokument výsledku. V našem konkrétním případě tak přeskočíme třetí vrácený výsledek, protože je

nerelevantní. Výpočet pak probíhá podle vzorce:

$$R(i) = \frac{Relevant\ hits}{Hits} \tag{9.3}$$

kde i je aktuálně zpracovaný dokument výsledku. Konkrétní dosazení do vzorců pak vypadá následovně:

$$\begin{aligned} R(1) &= \frac{1}{1} = 1 \\ R(2) &= \frac{2}{2} = 1 \\ R(4) &= \frac{3}{4} = 0,75 \end{aligned} \tag{9.4}$$

Samotná hodnota Average Precision je pak jen průměrem získaných hodnot:

$$AvrPrecision = \frac{\sum_{i=1}^{N}(P(r) \times rel(r))}{Number\ of\ relevant\ hits} \tag{9.5}$$

V našem případě:

$$AvrPrecision = \frac{R(1) + R(2) + R(4)}{3} = \frac{1 + 1 + 0,75}{3} = \frac{2,75}{3} = 0,916 \tag{9.6}$$

Pro tento konkrétní případ by tedy byla average precision rovna 0,916. Metrika MAP je pak průměrem všech average precisions získaných z každého testovaného dotazu.

$$MAP = \frac{Add\ all\ precision\ calculations\ together}{Total\ number\ of\ queries} \tag{9.7}$$

Takto dokážeme určit kvalitu vyhledávacího algoritmu v případě, že nelze vyhodnotit všechny vrácené výsledky dotazu. V této části bylo čerpáno z [12].

8.2 Vybrané metody a jednotlivé testy

Pro hodnocení kvality vyhledávání algoritmů jsem vybral metriku MAP, neboť jak jsem se zmínil výše, tato metrika zohledňuje umístění relevantních dokumentů (článků) ve výsledku. V testování jsem vyhodnocoval prvních 100 vrácených dokumentů každého dotazu. Samotné testování probíhalo nad 10 dotazy, které lze v původním stavu vidět v prvním sloupci tabulky 8.1.

Testování probíhalo ve třech různých variantách. V první byly testovány jednoduché výrazy, ve druhé variantě jsem zkoumal ovlivnění výsledku rozšířením výrazů o vybraná klíčová slova. Nakonec byl na sadě obohacené o klíčová slova testován vliv stop slov.

Pro výpočet jednotlivých hodnot byl vytvořen pomocný program, který očekává jako vstup soubour *testresult*, který obsahuje odřádkované hodnoty 0 a 1 (0=Nerelevantní 1=Relevantní). Tento soubor představuje ohodnocení relevantnosti výsledků jednoho dotazu. Samotný program pak po spuštění vypíše počet relevantních, nerelevantních a všech výsledků. Dále pak vypíše hodnotu precision a average precision. Zdrojové kódy tohoto programu se nachází na přiloženém CD v adresáři $source\_files/MAP\_count$.

8.2.1 Testování jednoduchých výrazů

V první variantě testů byly otestovány jednoduché výrazy. Výsledky jednotlivých dotazů lze pak vidět v tabulce 8.1. Ze získaných hodnot vyplývá, že ve většině případů je pro prvních 100 vrácených dokumentů hodnota mean average precision (dále jako MAP) nízká a tudíž také kvalita výsledku nedostačující. To je dle mého názoru způsobeno tím, že jednotlivé dotazy byly ve většině případů zadány příliš obecně. Tuto skutečnost jsem také ověřil v dalším testu (viz další podkapitola).

Obrázek 8.1: Testování algoritmu na dotazech

	10 vrácených výsledků			50 vrácených výsledků			100 vrácených výsledků		
dotaz	R	NR	AP	R	NR	AP	R	NR	AP
rybičky	0	10	0,000	2	48	0,043	4	96	0,044
kočkovité	7	3	0,948	28	22	0,664	41	59	0,618
Moto GP	4	6	0,307	30	20	0,484	34	66	0,485
hrady a zámky	9	1	0,947	41	9	0,891	76	24	0,839
tažní ptáci	7	3	0,948	36	14	0,741	54	46	0,692
RPG	3	7	0,643	11	39	0,364	19	81	0,293
plazi	3	7	0,338	19	31	0,393	38	62	0,381
telefony	8	2	0,916	29	21	0,658	45	55	0,611
pohádkové postavy	4	6	0,854	24	26	0,518	42	58	0,489
plži	9	1	0,989	26	24	0,789	51	49	0,661
mean average precision			**0,689**			**0,555**			**0,5113**

Popis: R .... relevantních dokumentů
NR .... nerelevantních dokumentů
AP ..... average precision

V tabulce lze dále vidět, že vyhodnocení relevantnosti bylo provedeno také pro prvních 10 a 50 vrácených dokumentů. Toto vyhodnocení jsem provedl, neboť jsem chtěl zjistit, zda se ve výsledku nachází více výše umístěných relevantních výsledků. Bylo tím zjištěno, že u všech dotazů byla hodnota MAP lepší v první polovině vrácených výsledků. Nejlepší hodnota average precision (dále jako AP) jednotlivých dotazů byla ve většině případů u prvních 10 vrácených stránek. Průměrná hodnota MAP z nich získaná byla 68,9%. S rostoucím počtem zpracovaných výsledků úměrně klesala hodnota MAP, kde pro 50 vrácených dokumentů byla její hodnota 55,5% a pro 100 vrácených dokumentů dokonce 51,13%.

8.2.2 Hodnocení vlivu použití klíčových slov

Jednou z hlavních implementovaných částí vyhledávacího algoritmu je výpočet klíčových slov, které se zobrazí uživateli. Jak jsem již popsal v kapitole 7, na základě těchto slov může uživatel upřesnit vyhledávání. V této testové variantě je tedy mým cílem otestovat změnu a případné vylepšení výsledku dotazu při použití těchto klíčových slov. Výsledky jednotlivých testů lze vidět v tabulce 8.2, kde v prvním sloupci jsou napsány jednotlivé výrazy obohacené o vybraná klíčová slova. Stejně jako v první variantě testů jsem zde vyhodnocoval relevantnost výsledků pro prvních 10, 50 a 100 vrácených dokumentů.

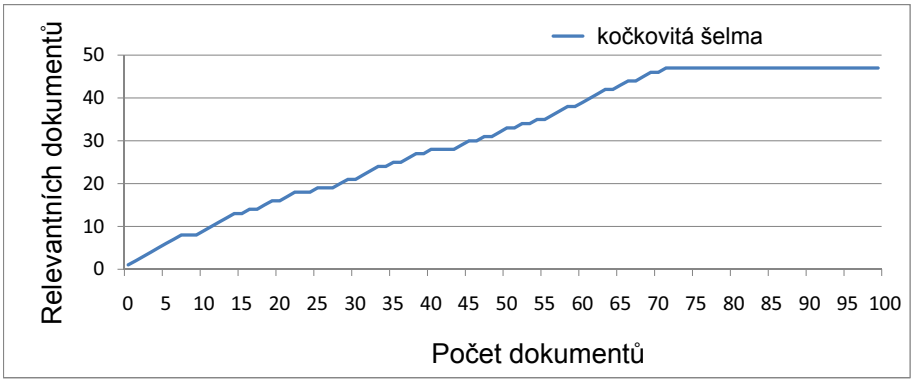
Ze získaných výsledků vyplývá, že po upřesnění jednotlivých výrazů pomocí klíčových slov se u vrácených dokumentů podstatně zlepšila metrika MAP. Její hodnota konkrétně dosahovala pro 100 vrácených dokumentů hodnoty 74,36%, pro 50 vrácených dokumentů 79,3% a pro 10 vrácených dokumentů dokonce 93,4%. Z těchto třech zmíněných hodnot také vyplývá, že se podstatná část relevantních dokumentů nacházela v horní polovině

Obrázek 8.2: Vliv použití klíčových slov

	10 vrácených výsledků			50 vrácených výsledků			100 vrácených výsledků		
dotaz	R	NR	AP	R	NR	AP	R	NR	AP
akvarijní ryby	5	5	0,927	26	24	0,671	40	60	0,641
kočkovitá šelma	8	2	1,000	32	18	0,804	47	53	0,754
Mistrovství světa silničních motocyklů Moto GP	10	0	1,000	31	19	0,891	55	45	0,851
hrady a zámky Čech	10	0	1,000	46	4	0,969	84	16	0,913
tažní stěhovaví ptáci	9	1	0,989	42	8	0,893	64	36	0,837
RPG hry na počítači	10	0	1,000	34	16	0,885	58	42	0,79
plazi a hadi	5	5	0,808	33	17	0,662	62	38	0,656
mobilní telefony	9	1	0,878	30	20	0,666	60	40	0,64
filmové pohádkové postavy	6	4	0,733	24	26	0,583	51	49	0,546
slimáci a šneci	10	0	1,000	37	13	0,901	66	34	0,808
mean average precision			**0,934**			**0,793**			**0,7436**

Popis: R .... relevantních dokumentů
NR .... nerelevantních dokumentů
AP ..... average precision

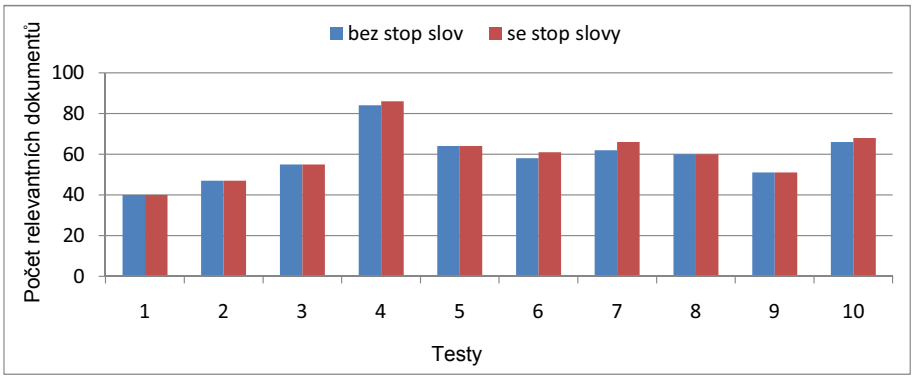
výsledku. Například pro dotaz 'kočkovitá šelma' se vyskytovala podstatná část relevantních dokumentů v prvních 70 vrácených dokumentech (viz obr 8.3), což je žádoucí.



Obrázek 8.3: Příklady výskytu relevantních dokumentů ve výsledku

8.2.3 Hodnocení vlivu použití stop slov

Sada testovaných dotazů obsahuje například výrazy 'RPG hry na počítači', či 'hrady a zámky Čech'. Dotazy v tomto tvaru obsahují kromě významových slov také slova, které pro vyhledávání nemají význam. Jde o tzv. stop slova, ze kterých je vytvořen seznam, na základě něhož jsou jednotlivé nevýznamová slova odstraněny z vyhledávání. Získání a vytvoření tohoto seznamu stop slov je popsáno v předešlé kapitole v sekci 7.4. Cílem této části testování je tedy vyhodnotit, jak velký vliv má na výsledek filtrování těchto slov pomocí seznamu stop slov.



Obrázek 8.4: Příklady porovnání vlivu stop slov

Z grafu 8.4 lze vidět, že odstranění stop slov nemá žádný negativní účinek na výsledek. Ve většině případů byl počet relevantních výsledků stejný. U některých testů se pak lišily výsledky minimálně.

8.3 Shrnutí a zhodnocení

V této kapitole jsem provedl hodnocení vyhledávacího algoritmu a vliv jednotlivých faktorů na výsledek vyhledávání. V první části jsem vyhodnotil kvalitu vyhledávacího algoritmu na sadě jednoduchých deseti výrazů. Z výsledků vyplynulo, že metrika MAP je zde pro prvních 100 vrácených dokumentů rovna 51,13%. Jak jsem se již zmínil výše, tato hodnota je ovlivněna obecnou formulací jednotlivých výrazů.

V další fázi byla testována změna kvality výsledku při použití klíčových slov jednotlivých článků. Bylo zjištěno, že použití těchto slov vylepšilo a upřesnilo obecně a nedostatečně formulované dotazy. Celková hodnota MAP zde pro prvních 100 vrácených dokumentů rovnala hodnotě 74,36%.

V poslední části byl testován vliv filtrování stop slov na kvalitu vyhledávání. Z výsledků vyplynulo, že filtrování stop slov nemá žádný negativní vliv na výsledek vyhledávání.

Kapitola 9

Závěr

Oblast vyhledávání informací je neustále ve vývoji. Každým dnem vzniká velké množství dat, které je nutné analyzovat, zpracovat, naindexovat a tím umožnit jejich kvalitní a rychlé vyhledání. Problémem vyhledávání v dnešní době je nalezení často nerelevantních dokumentů, což je ve většině případů způsobeno nepřesností přirozeného jazyka.

V této diplomové práci jsem nejdříve nastudoval oblast vyhledávání informací a její problematiku. Ta je podrobně rozebrána v druhé až čtvrté kapitole. V druhé kapitole je provedeno seznámení s problematikou vyhledávání informací a také jsou zde vysvětleny základní pojmy. Ve třetí kapitole jsou popsány a rozebrány klasické modely vyhledávání. Ve čtvrté kapitole je samotný výzkum zaměřen na textová data, sémantické slovníky a získávání znalostí odvoditelných z encyklopedií jako je Wikipedie. Na závěr této kapitoly jsou popsány a rozebrány jednotlivé nejznámější technologie pro fulltextové vyhledávání.

Problematiku vyhledávání dat v rámci interaktivně specifikovaných kontextů jsem rozebral v páté kapitole. Konkrétně jsem se zaměřil na řešení problematiky vyhledávání ve Wikipedii. Zde jsem po domluvě s vedoucím vybral a analyzoval možné využití nástroje Wikipedia Miner Toolkit.

V dalším kroku jsem provedl návrh vyhledávacího systému. Ten se skládá z více kroků: vyhledání slov ve slovníku, odstranění stop slov, provedení případného stemmování a vyhledávání dokumentů a jejich klíčových slov. Dalším krokem této práce byla samotná implementace vyhledávacího systému. Ten umožňuje provádět fulltextové vyhledávání. Jednotlivé kroky zprovoznění, implementace a problémy jsou popsány v kapitole 7. K implementovanému algoritmu je také vytvořena dokumentace, která je součástí přiloženého CD. Na závěr jsem provedl otestování kvality vyhledávače a vliv jednotlivých faktorů. Toto testování a jeho zhodnocení je popsáno v 8. kapitole.

Implementovaný vyhledávač umožňuje provádět kromě samotného vyhledávání také zobrazení jednotlivých klíčových slov odpovídajících článků. To umožňuje nejen v rychlosti pochopit obsah daného článku, ale také poskytuje možnost dalšího vyhledávání na základě těchto získaných slov. Další výhodou tohoto vyhledávacího systému je využití českého stemmeru od Ing. Davida Hellebranda. Ten umožňuje získat kořen slov, které se nenachází v dostupném slovníku českých slov.

Mezi další výhody implementovaného systému považuji možnost nastavení jednotlivých parametrů. Mezi ně patří nastavení počtu vrácených výsledků, úroveň příbuznosti klíčových slov, míra upřednostnění nebo určení zdroje vyhledávání (viz kapitola 8).

Jako možné rozšíření této práce by mohlo být zaměření se na sémantické dělení do kategorií, kde pro zadané výrazy by se uživateli zobrazil seznam možných kategorií, do kterých vyhledaný výraz významově spadá. Aktuálně je zde ale problémem nepříliš kvalitně

zprácované dělení do kategorií u české Wikipedie.

Literatura

[1] ABC Linuxu: *Text*. [cit. 2010-12-12], 2005.
URL http://www.abclinuxu.cz/ucebnice/zaklady/text

[2] Apache: *Apache Lucene*. 2010[cit. 2011-05-11].
URL http://lucene.apache.org/java/docs/index.html

[3] ARSLAN A., YILMAZEL O.: *Quality benchmarking Relational Databases and Lucene in the Trec4 Adhoc Task Enviroment*. 2001[cit. 2011-05-16].
URL
http://bildiri.anadolu.edu.tr/papers/bildirimakale/3219_b430w64.pdf

[4] BAEZA-YATES R.,RIBERIO-NETO B.: *Modern Information Retrieval*. ACM Press, 1999, iSBN-13: 978-0201398298.

[5] BARTÍK V.: *Dolování z textu a na webu*. FIT VUT Brno, 2008, přednáška k předmětu ZZN, Poslední modifikace 14.12.2010 [cit. 2010-12-12].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ZZN-IT/lectures/13_TextWeb.pdf

[6] BARTUNOV O., SIGAEV T.: *Tsearch2 - full text search extension for PostgreSQL*. [cit. 2011-03-08], 1991.
URL http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/

[7] FINKELSTEIN L.,GABRILOVICH E., MATIAS Y., RIVLIN E., SOLAN Z., WOLFMAN G., RUPPIN E.: *Placing Search in Context: The Concept Revisited*. [cit. 2011-05-22].
URL http://portal.acm.org/citation.cfm?id=313238.313279

[8] FUHR N.: *Probablistic models in information retrieval*. The computer Journal, 1992, 35(3):243-255.

[9] Fulltext search engines: In *Mediawiki*[online]. St. Petersburg (Florida) : Wikimedia Foundation,21.7.2005, Poslední modifikace 25.2.2011 [cit. 2010-05-11].
URL http://www.mediawiki.org/wiki/Fulltext_search_engines

[10] GROSSMAN D.A.,FRIEDER O.: *Information Retrieval*. Springer, 2004, iSBN 1-4020-3004-5.

[11] HELLEBRAND D.: *Nalezení slovních kořenů v češtině*. Ústav informačních systémů FIT VUT v Brně, 2010 [cit. 2011-04-22].
URL http://www.fit.vutbr.cz/study/DP/rpfile.php?id=7988

[12] LEE D.: *Performance Evaluationof Information Retrieval Systems*. 2008[cit. 2011-05-16].
URL http://www.scribd.com/doc/50445807/18/Mean-Average-Precision-MAP

[13] License, G. G. P.: *MnoGoSearch web search engine software*. 2000[cit. 2011-05-11].
URL http://www.mnogosearch.org/

[14] Manual:Importing XML dumps: In *Mediawiki*[online]. St. Petersburg (Florida) : Wikimedia Foundation,20.11.2008, Poslední modifikace 30.11.2010 [cit. 2010-12-12].
URL http://www.mediawiki.org/wiki/Manual:Importing_XML_dumps

[15] MEADOW CH.T. ,BOYCE B.R. a spol.: *Text Information Retrieval Systems*. Emerald Group Publishing, 2007.

[16] MediaWiki: In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation,7.11.2004, Poslední modifikace 28.1.2008 [cit. 2010-20-12].
URL http://cs.wikipedia.org/wiki/MediaWiki

[17] MediaWiki: *Obecně o MediaWiki*. [cit. 2010-12-12].
URL http://www.mediawiki.cz

[18] MILNE D.: An Open Source Toolkit for Mining Wikipedia. 2009[cit. 2010-12-12].
URL http://www.cs.waikato.ac.nz/~dnk2/publications/
\AnOpenSourceToolkitForMiningWikipedia.pdf

[19] NÉMETH L.: *Installation from Source Code*. [online], 2011, [cit. 2011-04-18].
URL http://hunspell.sourceforge.net/

[20] PORTER M.: *Snowball*. 2001[cit. 2011-05-16].
URL http://snowball.tartarus.org/

[21] PostgreSQL: *Text Search Functions and Operators*. 1996[cit. 2011-04-22].
URL
http://www.postgresql.org/docs/9.0/static/functions-textsearch.html

[22] PostgreSQL Global Development Group: *Installation from Source Code*. [online], 1996, [cit. 2011-04-18].
URL http://developer.postgresql.org/pgdocs/postgres/installation.html

[23] PostgreSQL Global Development Group: *PostgreSQL Core Distribution*. [online], 1996, [cit. 2011-04-18].
URL http://www.postgresql.org/download/

[24] PROSTŘEDNÍ P.: *Google Patent 20050071741: Podrobně*. 2005[cit. 2011-05-11].
URL http://www.businessweek.cz/google-patent-20050071741.html

[25] ROBERTSON S.E.,JONES K.S.: *Relevance weighting of search terms*. Journal of the American Society for Information Sciences, 1976, 27(3):129-146.

[26] SALTON G.,MCGILL M.J.: *Introduction to Modern Information Retrieval*. McGrawHill Book Co. New York, 1983.

[27] SINGHAL A.: *Modern Information Retrieval: A Brief Overview*. Google, Inc., 2001.
URL http://singhal.info/ieee2001.pdf

[28] Sémantika: In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation,17.6.2004, Poslední modifikace 20.5.2007 [cit. 2010-20-12]. URL http://cs.wikipedia.org/wiki/Sémantika

[29] VAN RIJSBERGEN C.J.: *Information Retrieval*. Butterworths, 1979.

[30] WebFinance, Inc.: Information. [online], [cit. 2010-21-11]. URL http://www.businessdictionary.com/definition/information.html

[31] Wikimedia - download: In *Mediawiki*[online]. St. Petersburg (Florida) : Wikimedia Foundation,20.5.2010, Poslední modifikace 5.5.2011 [cit. 2010-12-12]. URL http://www.mediawiki.org/wiki/Download

[32] Wikimedia Downloads: In *Mediawiki*[online]. St. Petersburg (Florida) : Wikimedia Foundation,20.5.2010, Poslední modifikace 5.5.2011 [cit. 2010-12-12]. URL http://dumps.wikimedia.org/

[33] Wikipedia: In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 7.9.2004, Poslední modifikace 7.9.2004 [cit. 2010-20-12]. URL http://cs.wikipedia.org/wiki/Wikipedia

[34] Wikipedia: In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation,21.11.2004, Poslední modifikace 19.7.2009 [cit. 2010-20-12]. URL http://cs.wikipedia.org/wiki/Wikimedia_Foundation

[35] YILMAZ E., KANOULAS E., ASLAM J.A.: *A Simple and Efficient Sampling Method for Estimating AP and NDCG*. Microsoft Research, College of Computer and Information Science, 2008.

[36] ZHANG Y.,XU YU J., HOU J.: *Web communities: analysis and construction*. Springer, 2006, iSBN 978-3-540-27737-8.

[37] ZUB O.: *Ispell/Aspell historie a současnost*. [online], 2003, [cit. 2010-04-18]. URL http://ozub81.sweb.cz/protokoly/tex/semestralka.pdf